SRP: a Scalable Resource Reservation Protocol for the Internet

Werner Almesberger <Werner.Almesberger@epfl.ch>[1],
Tiziana Ferrari <Tiziana.Ferrari@cnaf.infn.it>[2],
Jean-Yves Le Boudec <Leboudec@epfl.ch>[1]
[1]EPFL ICA, CH-1015 Lausanne, Switzerland
[2]DEIS, University of Bologna, and INFN/CNAF, Italy

March 2, 1998

Keywords: Resource reservation, Internet, Scalability, Bandwidth estimation

Abstract

Current resource reservation architectures for multimedia networks do not scale well for a large number of flows. We propose a new architecture that automatically aggregates flows on each link in the network. Therefore, the network has no knowledge of individual flows. There is no explicit signalling protocol, and the protocol overhead mainly consists in the introduction of a packet type with three values (reserved, request or best-effort) which can be encoded on two bits.

1 Introduction

Resource reservation architectures that have been proposed for integrated service networks (RSVP [1], ST-2 [2], Tenet [3], ATM [4, 5], etc.) all have in common that intermediate systems (routers or switches) need to store per-flow state information. This requirement probably stems from the desire to provide a network service that is as deterministic as possible. Now the IETF has proposed a class of reserved service, called *controlled load*, for which the assurance provided by the reservation is not absolute, but comparable to the one obtained by a best-effort stream in an unloaded network [6]. The controlled load service can be implemented through the accounting of the reservation of each individual stream, but not only: measurement based admission control methods [7] could be used as well. In the latter approach intermediate system functions could be simplified, since only aggregate flows need to be measured. On the contrary, in the former approach, and more generally in the current IETF resource reservation architecture, per individual flow processing leads to scalability problems when the number of individual flows increases. A solution is to aggregate flows inside the network, thus letting only the network edge be aware of all individual flows [8].

In this paper we propose a new way of performing reservations, which goes beyond concepts for aggregation on top of traditional reservation in that it makes aggregation the standard behavior of the network and not a special case requiring additional protocol activity. In short, our reservation model works as follows. A source that wishes to make a reservation (for example an adaptive multimedia application [9, 10]) starts by sending data packets marked with a *request* flag to the destination. These packets are forwarded normally by routers, who also take a flow admission decision on each of them. After enough *request* packets have been sent, the source learns from the destination its estimate of how much of the reservation has been accepted in the network. The source may then send data packets marked with a *reserved* flag at the accepted rate. Routers that have admitted, and thus forwarded, *request* packets have committed to have enough resources to accept subsequent *reserved* packets sent by the source at the accepted rate. The accepted rate is computed by every source, while routers independently estimate how much bandwidth they need to satisfy their global commitments. The accepted rate is guaranteed as long as there is a minimum activity by the source. As with the controlled load service, the guarantee is not absolute, but is only as good as the router estimator allows. The reservation disappears after the source has stayed idle for a while. The initial data packets sent by the source can be thought of as "sticky": once a router has accepted some of them at a given rate, it must continue to accept packets at the same rate until the source becomes idle.

Essential to our proposal is that routers do not keep state information per flow; routers only remember their reservation commitments globally per output port. This is made possible by two features:

- routers rely on end-systems not to exceed their accepted reservations or have to implement a mechanism which penalizes non-compliant sources;
- routers maintain reservations by learning, namely, by monitoring the actual reserved traffic and running an algorithm to estimate the bandwidth needed.

We discuss these two design directions in the rest of this section. Section 2 provides a protocol overview. Section 3 describes and compares different algorithms for the implementation of the traffic estimator and section 4 elaborates on that and also points out areas where more research is needed. Finally, protocol operation is illustrated with some simulation results in section 5 and the paper concludes with section 6.

1.1 Congestion control and reliance on end users

For best-effort traffic, the Internet has illustrated that network internals can be simple: besides routing, which has grown significant complexity, there are no "intelligent" services inside the network. Originally, congestion control (for example in TCP) has been entirely implemented in the end systems, which are in turn expected to have some degree of complexity of their own. Also, instead of providing stringent isolation among users, the Internet relies on guided cooperation. This approach is now being integrated with additional mechanisms to detect and penalize those non compliant sources which affect network performance [11].

Applying this approach to resource reservation means to let end systems perform flow acceptance control and to trust them not to exceed the agreed upon reservations. In order to protect the network from errors in application programs, the reservation protocol handling needs to be implemented in the networking kernel of the operating system.

If it is needed to counteract the abuses that might degrade network performance, then metering and policing functions can be implemented at network boundaries [12]. In a a typical example, policing would be performed at the boundary between a local area network and a public Internet service provider, as well as between Internet service providers. At each of these points, policing would be applied to aggregate flows, where an aggregate flow corresponds to a customer, or a group of customers, as for example in [13] or [14]. This is easily possible with the architecture proposed in this paper as we do not require any per flow information in the reservation mechanism in the routers. Policing and metering is outside the scope of this paper.

1.2 Learning by example

New reservations are set up by sending data packets with a *request* flag. When a router accepts such requests, it predicts the arrival of future packets and changes its global reservation state accordingly.

Because the reservation information is sent directly with the data, the reservation and the actual traffic are automatically synchronized.

Central to our proposal is the concept of estimation modules used by sources, routers, and destinations. In a simple implementation, we just count the number of request packets during a time interval and use it to predict the reserved bandwidth. This is the algorithm we propose to use at sources. Destinations also use this algorithm to compute the allowed rate sent back to the source. The situation for routers is more complex. In a simplistic solution, routers could use the same algorithm, with the difference that it is applied to aggregate flows. However this simplistic algorithm may in some case grossly over- or under-estimate the required resources for aggregate flows.

Flow multiplexing has a two-fold effect. On one hand it makes the aggregate traffic profile more stable, on the other one, in spite of the aggregation effect, the estimator in the router has to detect decreases in the use of reserved bandwidth (when sources reduce their rate or stop sending) and to adjust the reserved resources automatically. Therefore, we propose an alternative estimation algorithm for routers based on the computation of the "deterministic effective bandwidth" [15] and a feedback loop. We describe both implementations in section 3 and 5.1.

A more general discussion of measurement-based admission control for similar purposes can be found in [16]. Section 2 describes the components of our architecture, while 3 provides two algorithms for the implementation of the estimator. The evaluation of their performance is ongoing work.

2 Architecture overview

The proposed architecture uses two protocols to manage reservations: a reservation protocol to establish and maintain them, and a feedback protocol to inform the sender about the reservation status.

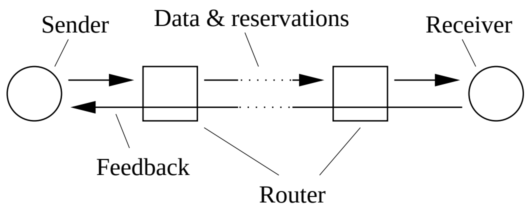

Figure 1: Overview of the components in SRP.

Figure 1 illustrates the operation of the two protocols:

- Data packets with reservation information are sent from the sender to the receiver. The reservation information consists in a packet type which can take three values and can thus be encoded on two bits. It is processed by routers, and may be modified by routers. Routers may also discard packets (section 2.1).
- The receiver sends feedback information back to the sender. Routers only forward this information; they don't need to process it (section 2.2).

Routers monitor the reserved traffic which is effectively present and adjust their global state information accordingly. Sections 2.1 and 2.2 illustrate the reservation and feedback protocol, respectively.

2.1 Reservation protocol

The reservation protocol is used in the direction from the sender to the receiver. It is implemented by the sender, the receiver, and the routers between them. As mentioned earlier, the reservation information is a packet type which may take three values:[1]

Request This packet is part of a flow which is trying to gain *reserved* status. Routers may accept, degrade or reject such packets. When routers accept some request packets, then they commit to accept in the future a flow of reserved packets at the same rate. The exact definition of the rate is part of the estimator module.

Reserved This label identifies packets which are inside the source's profile and are allowed to make use of the reservation previously established by *request* packets. Given a correct estimation, routers should never discard *reserved* packets because of resource shortage.

Best effort No reservation is attempted by this packet.

Packet types are initially assigned by the sender, as shown in figure 2. A traffic source (i.e. the application) specifies for each packet if that packet needs a reservation. If no reservation is necessary, the packet is simply sent as *best-effort*. If a reservation is needed, the protocol entity checks if an already established reservation at the source covers the current packet. If so, the packet is sent as *reserved*, otherwise an additional reservation is requested by sending the packet as *request*.

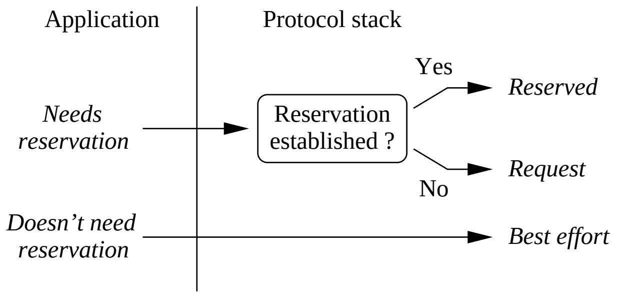

Figure 2: Initial packet type assignment by sender.

Each router performs two processing steps (see also figure 3). First, for each *request* and *reserved* packet the estimator updates its current estimate of the resources used by the aggregate flows and decides whether to accept the packet (packet admission control). Then, packets are processed by various schedulers and queue managers inside the router.

- When a *reserved* packet is received, the estimator updates the resource estimation. The packet is automatically forwarded unchanged to the scheduler where it will have priority over best-effort traffic and normally is not discarded.
- When a *request* packet is received, then the estimator checks whether accepting the packet will not exceed the available resources. If the packet can be accepted, its *request* label is not modified. If the packet cannot be accepted, then it is degraded to best-effort
- If a scheduler or queue manager cannot accept a *reserved* or *request* packet, then the packet is either discarded or downgraded to *best-effort*.

Note that the reservation protocol may "tunnel" through routers that don't implement reservations. This allows the use of unmodified equipment in parts of the network which are dimensioned such that congestion is not a problem.

The receiver does no packet-type specific processing. Instead, it counts incoming packets and sends feedback to the sources.

[1] The encoding is yet unspecified; many possibilities exist, such as: encodings being defined by the differentiated services working group at IETF, the class of service bits in the MPLS label, or a new IP option.

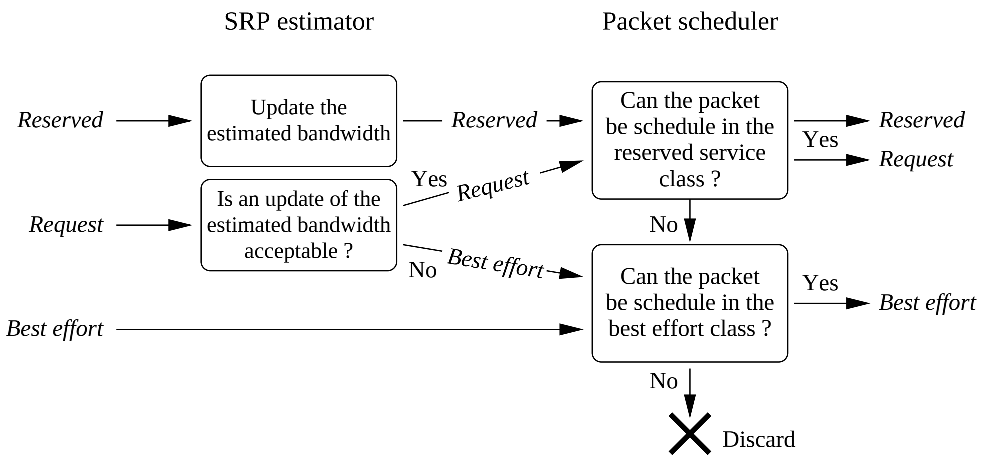
SRP estimator
Packet scheduler
Reserved
Update the estimated bandwidth
Reserved
Can the packet be schedule in the reserved service class ?
Yes
Reserved
Request
Request
Is an update of the estimated bandwidth acceptable ?
Yes
Request
No
Best effort
No
Can the packet be schedule in the best effort class ?
Yes
Best effort
Best effort
No
Discard

Figure 3: Packet processing by routers.

2.2 Feedback protocol

The feedback protocol is used to convey information on the success of reservations and on the network status from the receiver to the sender. Unlike the reservation protocol, the feedback protocol does not need to be interpreted by routers, because they can determine the reservation status from the sender's choice of packet types.

Feedback information is collected by the receiver and it is periodically sent to the sender. The feedback consists of the receiver's estimate of the current reservation. The receiver computes this estimate with its local estimator. Additional information can be included in feedback messages to improve stability and to provide additional information on network performance, e.g. the loss rate along the path or the round-trip time.

The reservation estimated by the receiver is an upper bound for the rate at which the sender may send reserved traffic and is used by the sender to decide whether packets are sent as *reserved* or *request*. The other upper bound is provided at the source by the local estimate of the requested rate, so the maximum rate at which the source is allowed to send *reserved* packets is $\min(feedback, src\_estimate)$.

Receivers collect feedback information independently for each sender and senders maintain the reservation state independently for each receiver. Note that, if more than one flow to the same destination exists, attribution of reservations is a local decision at the source.

The feedback mechanism can be implemented on top of a protocol like RTCP [17].

Figure 4 provides the overall picture of the reservation and feedback protocols for two end-systems connected through routers $R1$ and $R2$. The initial resource acquisition phase is followed by the generation of request packets after the first feedback message arrives. Dotted arrows correspond to degraded *request* packets, which passed the admission control test at router $R1$ but could not be accepted at router $R2$ because of resource shortage. Degradation of *request*s is taken into account by the feedback protocol. After receiving the feedback information the source sends *reserved* packets at an appropriate rate, which is smaller than the one at which *request* packets were generated. At point 1 router $R1$ starts accounting *request* packets, while at point 2 *reserved* packets take the place of *request*s as a consequence of the feedback received by the source.

3 Estimation modules

We call *estimator* the algorithm which attempts to calculate the amount of resources that need to be reserved. The estimation measures the number of *request*s sent by sources and the number of *reserved* packets which actually make use of the reservation.

Estimators are used for several functions:

- Senders use the estimator for an optimistic prediction of the reservation the network will perform for the traffic they emit. This, in conjunction with feedback received from the receiver, is used to decide whether to send *request* or *reserved* packets.

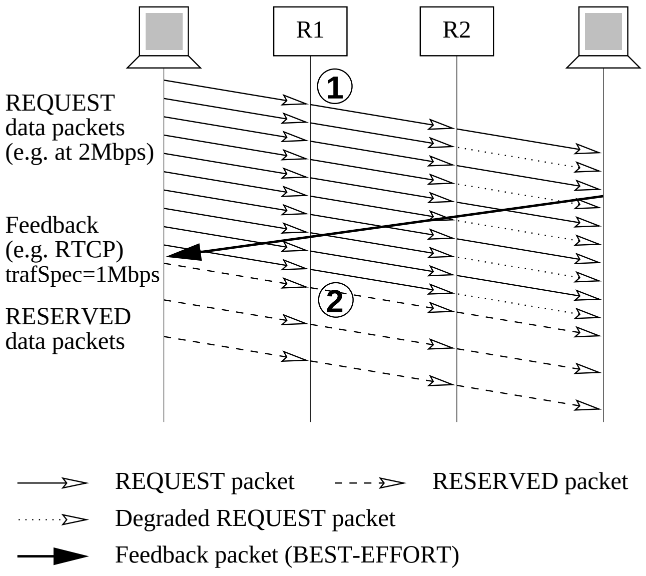

Figure 4: Reservation and feedback protocol diagram.

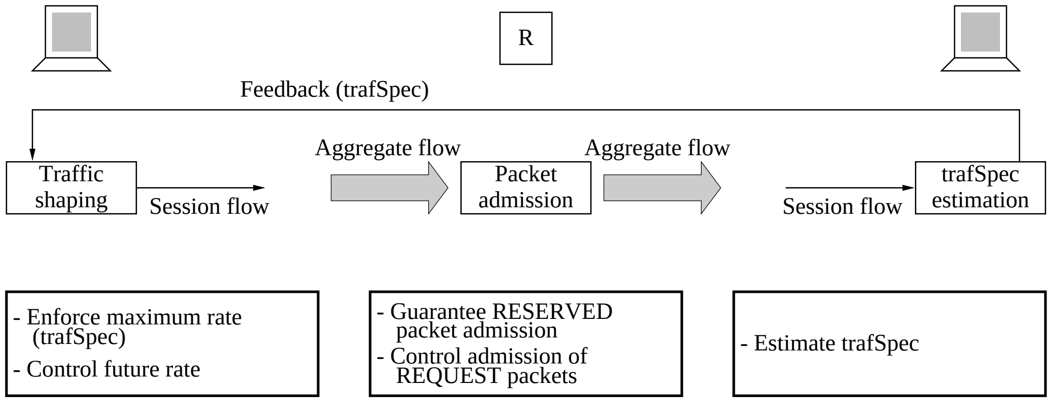

Figure 5: Use of estimators at senders, routers, and receivers

- Routers use the estimator for packet-wise admission control and perhaps also to detect anomalies (see section 4.4).
- In receivers, the estimator is fed with the received traffic and it generates a (conservative) estimate of the reservation at the last router. This is sent as feedback to the source.

Figure 5 shows how the estimator algorithm is used in all network elements.

First, a source estimates the reservation established by its traffic. This is an *optimistic* estimation since it assumes that all *request*s are accepted. Then, it collects the feedback information, which is *conservative*. In fact this parameter is the estimation from the point of view of the last hop router, which doesn't take into account *request* and/or *reserved* packets which may have been dropped on the path (the estimation could be higher on other intermediate systems). A source always uses the minimum of these two parameters to determine the right output traffic profile.

As described in section 2.1, a sender keeps on sending requests until successful reservation setup is indicated with a feedback packet. This means that the sender sends more requests than needed if the round-trip-time is greater than the interval necessary to allocate resources, since in this way the source keeps sending *request*s even after the desired amount of resources has been reserved. It's the feedback that is returned to the sender, which indicates the right allocation obtained on the path. Since the source is feedback-compliant after an interval the routers on the path start releasing a part of the over-estimated reservation already allocated. The feedback that is returned to the sender may also show an increased number of requests. The sender must not interpret those requests as a direct increase of the reservation, because the routers didn't either. Instead, the sender estimator must correct the feedback information accordingly, which is achieved through the computation of the minimum of the feedback and of the resource amount requested by the source.

Our architecture is independent of the specific algorithm used to implement the estimator. Sections 3.1 and 3.2 describe two different solutions. The definition and evaluation of algorithms for reservation calculation in hosts and routers is still ongoing work.

3.1 Basic estimation algorithm

The basic algorithm we present here is suitable for sources and destinations, and could be used as a rough estimator by routers. This estimator counts the number of requests it receives (and accepts) during a certain *observation interval* and use this as an estimate for the bandwidth that will be used in future intervals of the same duration.

In addition to requests for new reservations, the use of existing reservations needs to be measured too. This way, reservations of sources that stop sending or that decrease their sending rate can automatically be removed. For this purpose the use of reservations can be simply measured by counting the number of *reserved* packets that are received in a certain interval.

To compensate for deviations caused by delay variations, spurious packet loss (e.g. in a best-effort part of the network), etc., reservations can be "held" for more than one observation interval. This can be accomplished by remembering the observed traffic over several intervals and using the maximum of these values (step 3 of the following algorithm). Given a hold time of h observation intervals, the maximum amount of resources which can be allocated Max, res and req (the total number of *reserved* and *request bytes* received in a given observation interval), the reservation R (in bytes) is computed by a router as follows. Given a packet of n bytes:

```
if (packet_type == REQ)
    if (R + req + n < Max) {
        accept;
        req = req + n;   // step 1
    }
    else degrade;

if (packet_type == RES)
    if (res + n < R) {
        accept;
        res = res + n;   // step 2
    }
    else degrade;
```

where initially $R, res, req = 0$. At the end of each observation cycle the following steps are computed:

```
i = h;
while (i > 1) {
    R[i] = R[i-1];
    i--;
}
R[1] = res + req;
R = max (R[h], R[h-1], ..., R[1]);  // step 3
res = req = 0;
```

The same algorithm is run by the destination with the only difference that no admission checks are needed. Examples of the operation of the basic algorithm are shown in section 5.1.

This easy algorithm presents several problems. First of all, the choice of the right value of the observation interval is critical and difficult. Small values make the estimation dependent on bursts of *reserved* or *request* packets and cause an overestimation of the resources needed. On the other hand, large intervals make the estimator react slowly to changes in the traffic profile. Then, the strictness of traffic acceptance control is fixed, while adaptivity would be highly desirable in order to make the allocation of new resources stricter as the amount of resources reserved gets closer to the maximum. These problems can be solved by devising an adaptive enhanced algorithm like the one described in the following section.

3.2 Enhanced estimation algorithm

Instead of using the same estimator in every network component, we can enhance the previous approach so that senders and receivers still run the simple algorithm described above, while routers implement the following modified estimator.

There are two components in this method: the admission control based on a low pass filter and the admission control adaption, which uses a feedback based on the observed performance of prior estimates, e.g. on the rate of rejected *reserved* and *request* packets estimated by the router.

Packet admission control and low pass filter In order to filter out small scale traffic profile variations in a way close to the real node behavior, we borrow the concept of *deterministic effective bandwidth* from network calculus [15]. Given an arrival curve α and a delay bound D, the corresponding deterministic effective bandwidth e_D is defined as:

$$e_D = \sup_{s \geq 0} \frac{\alpha(s)}{s + D}$$

By applying this definition to our model and by assuming that observation starts at time 0, we obtain that:

$$e = \sup_{1 \leq i \leq j} \frac{n_i + \ldots + n_j}{t_j - t_i + D}$$

where $t_1, \ldots, t_k$ are the time instants at which packets arrive, n_i is the number of bytes in packet number i (only *reserved* or *request* packets are taken into account) and D is a fixed parameter: the delay objective.

e represents the bandwidth required for the flow with smoothed peaks, as packets are queued in a buffer system requesting a maximum queueing time of D. Since the traffic profile of a flow may change, the capacity estimated for a given flow should vary accordingly. To achieve this we estimate the effective bandwidth e at any arrival time t_k of a *reserved* or *request* packet over a sliding window w:

$$e_k = \sup_{1 \leq i \leq j \text{ and } t_i, t_j \in [t_k - w, t_k]} \frac{n_i + \ldots + n_j}{t_j - t_i + D} \tag{1}$$

Then, in order to smooth out changes in e_k as a function of the packet rate of a flow we eventually calculate γ by taking the exponentially weighted average of e_k and we assume that the amount of bandwidth allocated by a router at time t_k per input and output port is equal to γ:

$$\gamma_k = \alpha^d \gamma_{k-1} + \left(1 - \alpha^d\right) e_k \tag{2}$$

where α is a parameter such that $0 < \alpha < 1$, and $d = t_k - t_{k-1}$. α^d is the weight, which depends on the time between packets. If no packets arrive for a long time, then $\gamma_k \approx e_k$ as expected and if e is constant, then γ_k converges towards e, with a speed depending on α. Parameters α and w define the behavior of the low pass filter, in particular the resource release process of the estimator when a given flow stops, and the reservation keeping during temporary silences.

The packet admission procedure is devised in such a way that *reserved packets* are always considered in the estimator, while *request* packets have to pass an admission control test. If the *k-th* packet is *reserved*, then equations 1 and 2 are computed and the packet is accepted, even if it could be discarded later by the scheduler. On the other hand, if the packet type is *request*, the following test is applied:

$$\text{if } \gamma_k \beta \leq C_{max} \text{ then accept else refuse}$$

where C_{max} [2] is a fixed parameter representing the maximum amount of bandwidth which can be reserved on a given output interface, and β is a correction factor computed according to the algorithm presented in the

[2]Rate C_{max} is enforced through a scheduler associated to a shaper.

following paragraph. If the packet is accepted then the estimated bandwidth is updated, otherwise the packet is downgraded to *best-effort* and we let $\gamma_k = \gamma_{k-1}$ (i.e. if a packet is rejected by the admission test, its arrival is ignored by the estimator).

Adaptivity in packet admission control Adaptivity in packet admission control is obtained by making parameter β vary as a function of the number of reserved bytes lost. There are two independent variables: L_r, the number of reserved bytes *really* lost by the router, and L_v, the number of reserved bytes *virtually* lost as defined in formula 3.

L_r is the measure of real losses of *reserved* and (accepted) *request* packet, which occur when the amount of reserved traffic reaches the capacity C_{max}. We assume that L_r is counted over intervals $(t-\theta, t]$ (see below).[3]

In order to tune β before *reserved* traffic reaches the capacity C_{max} the following parameter L_v is calculated at each packet arrival:

$$L_v := \max\bigl(0, L_v + n_k - \gamma_{k-1}\beta(t_k - t_{k-1})\bigr) \tag{3}$$

where n_k is the size of the current packet, $t_k - t_{k-1}$ is the time since the previous packet was received, and γ_{k-1} is the value of γ computed after the last packet reception. L_v represents the maximum buffer occupancy, counted in bytes, of a queue. We suppose that at time $t = 0$ this queue is empty and it is continuously served at rate $\gamma\beta$, which is the current estimate of the bandwidth required by the flow.

We obtain the maximum virtual queue size L_v^{max} by calculating

$$L_v^{max} := \max(L_v^{max}, L_v) \tag{4}$$

after updating L_v. The initial value of L_v^{max} is 0.

If our estimation procedure is correct, we should have $L_v^{max} \leq \gamma\beta D$, otherwise we need to increase the value of β. Conversely, if L_v^{max} is very small, then we have to decrease β.

To determine how to change β, we use L_v^{max} to calculate the rate $\gamma\beta'$ at which we have to serve the virtual queue to reach the length corresponding to the delay goal D at the present rate $\gamma\beta$:

$$\gamma\beta' = \gamma\beta + \frac{L_v^{max} - \gamma\beta D}{B^{-1}}$$

or

$$\beta' = \beta + B\left(\frac{L_v^{max}}{\gamma} - \beta D\right)$$

where B^{-1} is the time after which the length goal should be reached. β is updated with period θ as follows:

$$\beta := \beta \underbrace{+A\frac{L_r}{N_r}}_{\text{if } L_r>0} +B\left(\frac{L_v^{max}}{\gamma} - \beta D\right) \tag{5}$$

where N_r is the amount of data received in *reserved* and (accepted) *request* packets since the last update of β, L_r is the amount of such data lost in the same interval, γ is the current bandwidth estimate, and A and B are fixed parameters to be tuned by simulation. The initial value of β is 1. L_v^{max} is reset to L_v after computing (5).

The possibility to make β a function of the rejection rate of *request* packets and the tuning of the parameters used in the algorithms described above, are arguments for future work.

Examples of the operation of this algorithm are shown in section 5.2.

4 Additional aspects

This section describes further details of the proposed reservation architecture and discusses areas requiring further research.

[3]The actual measurement and filtering method for L_r is argument of further study.

4.1 Resource reservation in a router

This section gives an example of how resource reservation can be handled in a simple router where only output buffer space is controlled. Depending on its architecture, a real router may have to take the status and utilization of many other components into account.

Figure 6 illustrates the packet processing in the example router: After passing the router fabric, the reservation information in each packet is processed (see section 2.1). Packets of type *request* or *reserved* are put into the queue for reserved traffic. All other packets are put into the best-effort queue or they are discarded. The queues are emptied by a scheduler which gives priority to the reserved traffic queue. The scheduler may be more complex if sophisticated forms of link sharing are employed.

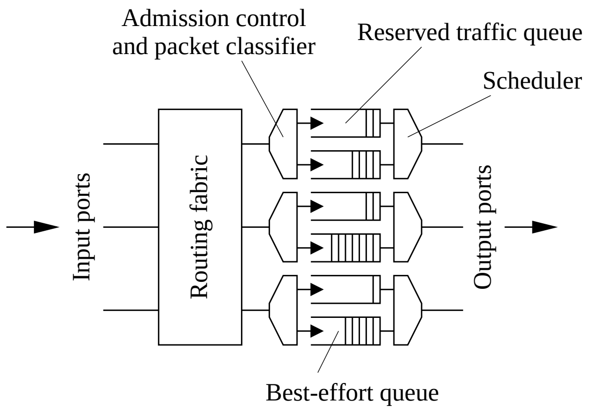

Figure 6: Example router.

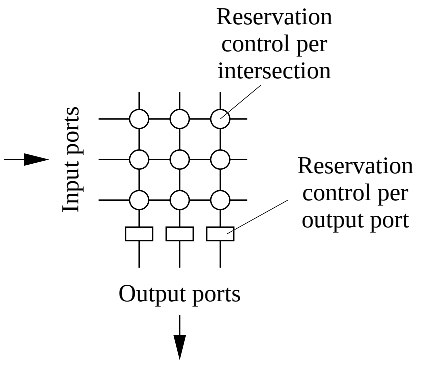
Reservation control per intersection
Input ports
Reservation control per output port
Output ports

Figure 7: Reservation control in router.

Placing the estimator directly before the output queues naturally leads to aggregation: since the critical resource at this point is queue space, one can for instance express reservations as allocations of such space within a given interval. The sum of the allocations then corresponds to the aggregate bandwidth, which is reserved on that port.

Detection of malfunction can be improved without impacting scalability by calculating reservations not only for each output port, but for each input and output port pair (which is called an "intersection" in figure 7).

4.2 Starvation

Reservation establishment is incremental. It is therefore possible for a sender to obtain only a fraction of the required resources if a shortage occurs before all the requests have been accepted. This can lead to starvation if several senders (unsuccessfully) compete for the same resource for an extended period of time.

A sender can react to this situation in the following ways:

- give up and report reservation failure to the application
- try to proceed with the partial reservation (e.g. if the shortage occurred during an attempt to increase an older reservation)
- back off and try again later

In the latter case, the sender has to wait for the hold time plus a random delay before sending new requests. The random delay should be exponentially increased on repeated reservation failures to the same destination.

4.3 Generation of inelastic best-effort traffic

Degrading *request* packets to *best-effort* during resource shortage is desirable, because it allows the communicating hosts to easily distinguish a mere reservation failure from a total communication breakdown.

Unfortunately, blindly converting all *request* packets to *best-effort* may have disastrous effects on other best-effort traffic: since a sender emits requests at the full rate of the desired reservation, the resulting inelastic best-effort traffic would be grossly unfair with respect to protocols like TCP, which perform end-to-end congestion control (see also [18]).

If the network implements a packet type for inelastic best-effort traffic[4] or generally a lower priority type than normal best-effort traffic, that type should be used when degrading *request* packets. Otherwise, a more aggressive discard policy has to be used for those packets. This could for instance be modulated by measuring congestion-controlled traffic (e.g. TCP) flowing to the same destination.

4.4 Route changes

Like most other reservation architectures, the proposed one may fail to provide the promised service if there is a route change. Architectural means to reduce the number of route changes to the absolutely necessary minimum (e.g. "route pinning" [19]) are outside the scope of this paper.

Once a route change occurs, e.g. due to a link failure, it typically has the following effects: The traffic is redirected to a path on which no prior reservation exists (**b**,**c**). Note that **A** cannot distinguish older reserved traffic sharing the path via **a** and **b** from redirected traffic and that it may therefore degrade *reserved* packets of the former.

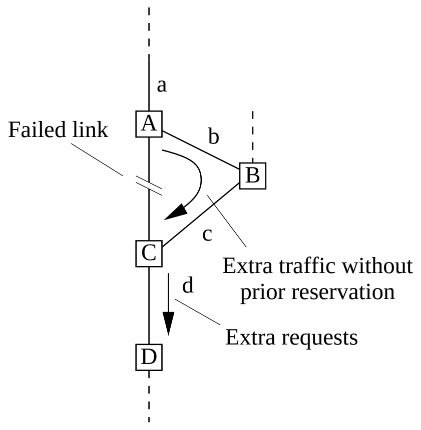

Figure 8: Route change example.

A further anomaly can occur if the original path and the redirected path merge again further downstream (**d**): The original reservation and the new requests that were generated to repair the reservation can collide and yield an artificially high reservation. This is similar to the time-to-feedback problem discussed in section 3 and the same mechanisms can be used to overcome it.

4.5 Multicast

The reservation mechanism described can be slightly extended to a multicast scenario. The extensions concern the feedback and the reservation protocol at the source. They are needed to cope with several problems which are typical in a multicast environment:

- the joining mechanism: how to establish reservations to a new group member without affecting the reservation already in place;
- transparency: events like route instability, topology changes, joining and leaving of some group members and situations like heterogeneous connectivity should only affect their limited scope. They should be completely transparent to the remaining session members and also to the connections established by other applications.
- feedback implosion: the feedback protocol which works well in a unicast scenario does not scale well in a multicast environment.

Establishing reservations in a multicast tree The mechanism described here to build up reservations in a multicast context fits for multicast routing algorithms based on the *Core Based Tree* approach [20, 21], in which sources do not flood traffic periodically to the network. In this way reservations (*request* and *reserved* packets) can be restricted to the links belonging to the multicast tree.

[4]Such a type would for instance have service characteristics like a low delay but a higher loss probability.

The source starts sending request messages to the multicast routers which explicitly joined the group as a reply to the source register message. Members of a session are divided into two sets:

1. joining members, forming the *request* multicast group;
2. "old" members, forming the *reserved* multicast group.

This distinction is necessary in order to make the joining operation transparent to the hosts and to the branches already belonging to the session.[5] The purpose of this division is to forward *request* packets only on the path from the nearest multicast router belonging to the reserved group to the new member, as shown in figure 9.

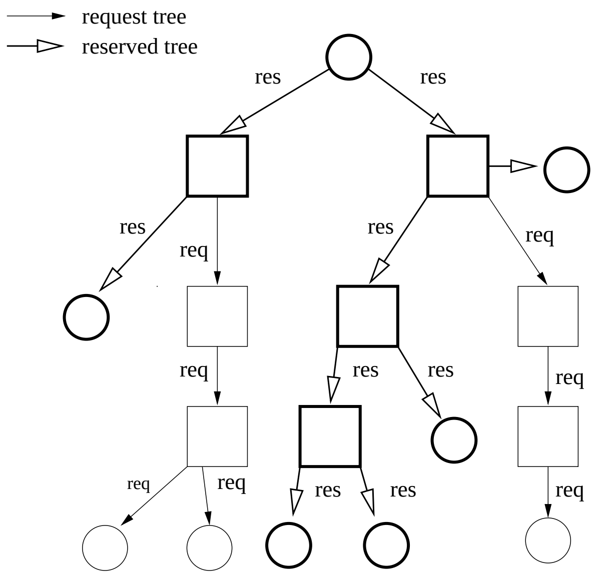

Figure 9: Request and reserved multicast group.

The join request is issued hop-by-hop toward a multicast router already on the reserved tree. Routers already receiving reserved traffic start sending the multicast traffic to the member after receiving the join request. In addition to that, they also switch the *reserved* flag to *request*. Members of the request group can compare their reservation estimate to the target amount indicated by the source. If the reservation offered is acceptable, then the member can leave the request group and join the reserved group.[6]

This mechanism can be implemented by associating two multicast addresses to the two distinct groups. The addresses can be different only in the least significant bit – for example it can be 0 for the request group and 1 for the reserved group. Then, the algorithm executed by the multicast router when a multicast packet is received, is the following:

```
if ((packet_addr is multicast) and (packet_type == RES)) {
    forward packet to reserved group;
    if (router is in the request group) {
        newpacket = copy(packet);
        newpacket_type = REQ;
        forward newpacket to request group;
    }
}
```

[5]New members cannot join directly the reserved group because this would have the effect of injecting *reserved* packets into links on which the corresponding amount of resources was not allocated before. Since routers have no means to distinguish "legal" from "illegal" packets, non-conforming data would affect other reservations already in place. Vice versa, the sending of *request* packets would have the effect of increasing the reservation level on the trunks already belonging to the reserved tree.

[6]This mechanism requires that the interval between the leaving and the joining is small compared to the life time of the reservation just established.

Transparency In a network with bottlenecks the algorithm should avoid that the link with worst connectivity (e.g. with the lowest bandwidth availability) limits the reservation offered to each member of the group. To cope with this heterogeneity multicast members could be grouped into separate sets and layered coding [22] could be used.

Different coding layers representing different levels of quality are sent to different multicast groups. All the receivers are included in a common multicast tree for the distribution of the fundamental coding layer, then other coding layers can be added to it. The traffic distribution of each layer can be implemented through the reserved and request group described above and each member can join several groups at the same time depending on the quality of its connectivity.

Feedback The problem of feedback implosion is solved by simply not sending any explicit feedback but by using group membership as an implicit indicator instead. The multicast source can fix an a priori value for the minimum amount of reservations required to forward the traffic of a given coding layer. After joining the request group the receiver does flow acceptance control. If the estimated reservation is acceptable compared to the target set by the source, then it can leave the *request* group and join the *reserved*, otherwise it leaves the *request* group and gives up. So, the absence of a reserved group of a session can mean two things: no members have joined the request group yet or no members can accept the reservation offered.

Since the source does not receive any feedback, it can statically fix the reservation threshold of each multicast group. If that amount of resources can not be allocated, hosts will leave both groups, the multicast trees disappear and the (partial) reservations time out.

5 Simulation

Section 5.1 provides a theoretic description of the behavior of the reservation mechanism in a very simple example, while section 5.2 shows the simulated behavior of the proposed architecture.

5.1 Reservation example

The network we use to illustrate the operation of the reservation mechanism, is shown in figure 10: the sender sends over a delay-less link to the router, which performs the reservation and forwards the traffic over a link with a delay of two time units to the receiver. The receiver periodically returns feedback to the sender.

The sender and the receiver both use the basic estimator algorithm described in section 3.1. The router may – and typically will – use a different algorithm (e.g. the one described in section 3.2).

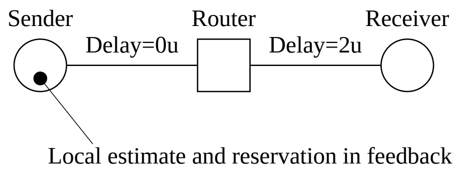

Figure 10: Example network configuration.

The bandwidth estimate at the source and the reservation that has been acknowledged in a feedback message from the receiver are measured. In figure 11, they are shown with a thin continuous line and a thick dashed line, respectively. The packets emitted by the source are indicated by arrows on the reservation line. A full arrow head corresponds to *request* packets, an empty arrow head corresponds to *reserved* packets. For simplicity, the sender and the receiver use exactly the same observation interval in this example, and the feedback rate is constant.

The source sends one packet per time unit. First, the source can only send requests and the router reserves some resources for each of them. At point (1), the estimator discovers that it has established a reservation for six packets in four time units, but that the source has only sent four packets in this interval. Therefore, it corrects its estimate and proceeds. The first feedback message reaches the sender at point (2). It indicates a reservation level of five packets in four time units (i.e. the estimate at the receiver at the time when the feedback was sent), so the sender can now send *reserved* packets instead of *requests*. At point (3), the next observation interval ends and the estimate is corrected once more. Finally, the second feedback arrives at point (4), indicating the final rate of four packets in four time units. The reservation does not change after that.

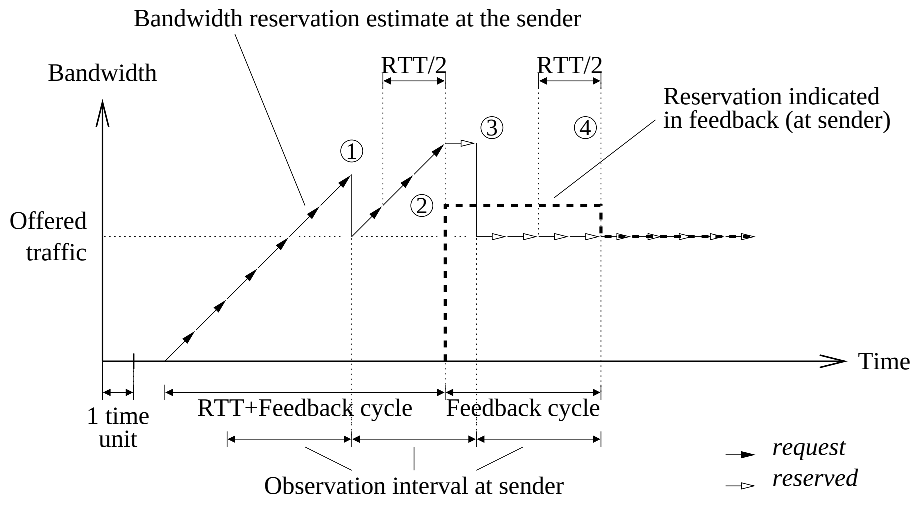
Bandwidth reservation estimate at the sender
Bandwidth
RTT/2
RTT/2
Reservation indicated in feedback (at sender)
①
②
③
④
Offered traffic
Time
1 time unit
RTT+Feedback cycle
Feedback cycle
Observation interval at sender
request
reserved

Figure 11: Basic estimator example.

5.2 Simulation results

The network configuration used for the simulation is shown in figure 12.[7] The grey paths mark flows we examine below.

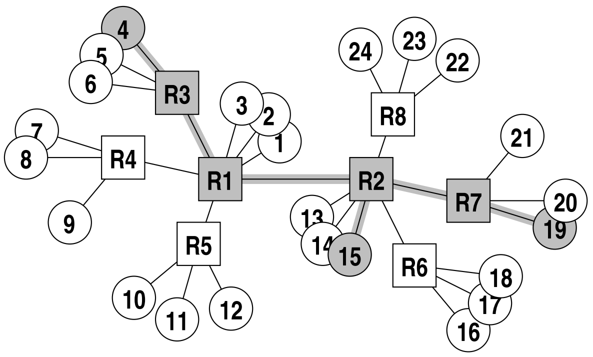

Figure 12: Configuration of the simulated network.

There are eight routers (labeled **R1**...**R8**) and 24 hosts (labeled **1**...**24**). Each of the hosts **1**...**12** tries occasionally to send to any of the hosts **13**...**24**. Connection parameters are chosen such that the average number of concurrently active sources sending via the **R1**–**R2** link is approximately fifty. The bandwidth of a flow remains constant while the flow is active and is chosen randomly from the interval $[1, 200]$.

All links in the network have a bandwidth of 4000 packets per second and a delay of 15 ms.[8] We allow up to 90% of the link capacity to be allocated to reserved traffic. The link between **R1** and **R2** is a bottleneck, which can only handle about 72% of the offered traffic. The delay objective D of each queue is 10 ms. The queue size per link is limited to 75 packets.

Figure 13 shows the **R1**–**R2** link as seen from **R1**. We show the total offered rate, the estimated reservation ($\gamma\beta$) and the smoothed actual rates of *request* and *reserved* packets.

Figure 14 shows the behaviour of the real queue. The system succeeds in limiting queuing delays to approximately the delay goal of 10 ms, which corresponds to a queue size of 40 packets. The queue limit of 75 packets is never reached.

[7]The programs and configuration files used for the simulation will be made available on `http://lrcwww.epfl.ch/srp/`

[8]Small random variations were added to link bandwidth and delay to avoid the entire network from being perfectly synchronized.

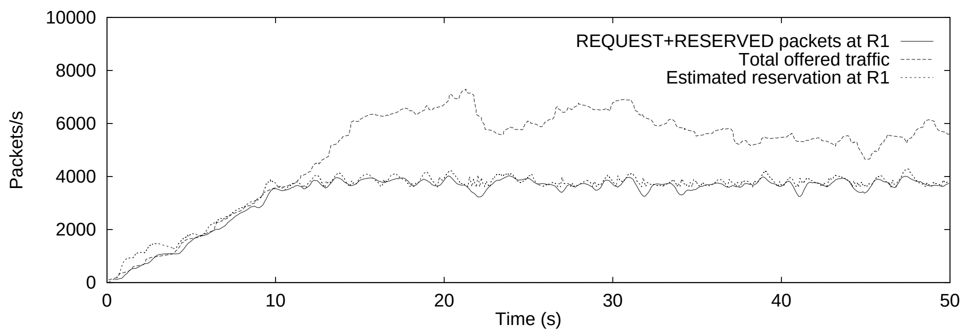

Figure 13: Estimation and actual traffic at **R1** towards **R2**.

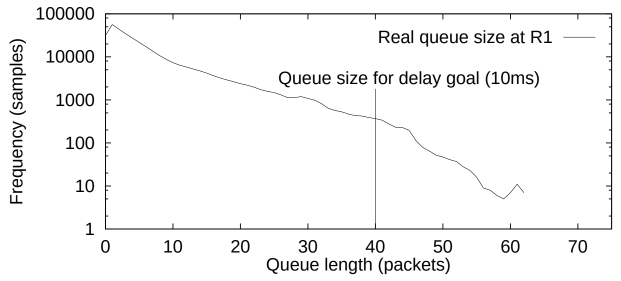

Figure 14: Queue length at **R1** on the link towards **R2**.

Finally, we examine some end-to-end flows. Figure 15 shows a successful reservation of 84 packets per second from host **4** to **15**. The requested rate, the estimation at the destination, and the (smoothed) rate of *reserved* packets are shown. Similarly, figure 16 shows the same data for a less successful reservation host **4** attempts later to **19**.

During the entire simulated interval of 50 seconds, 3'368 *request* packets and 164'723 *reserved* packets were sent from **R1** to **R2**. This is 83% of the bandwidth of that link.

6 Conclusion

We have proposed a new scalable resource reservation architecture for the Internet. Our architecture achieves scalability for a large number of concurrent flows by aggregating flows at each link. This aggregation is made possible by entrusting traffic control decisions to end systems – an idea borrowed from TCP. Reservations are controlled with estimation algorithms, which predict future resource usage based on previously observed traffic. Furthermore, protocol processing is simplified by attaching the reservation control information directly to data packets.

We did not present a conclusive specification but rather described the general concepts, gave examples for implementations of core elements, proposed estimator algorithms for sources, destinations and routers, and showed some illustrative simulation results. Further work is needed for a comprehensive specification, and future research will focus on evaluating and improving the estimator algorithms described in this paper, and on related algorithms.

References

[1] RFC2205; Braden, Bob (Ed.); Zhang, Lixia; Berson, Steve; Herzog, Shai; Jamin, Sugih. *Resource ReSerVation Protocol (RSVP) – Version 1 Functional Specification*, IETF, September 1997.

[2] RFC1819; Delgrossi, Luca; Berger, Louis. *ST2+ Protocol Specification*, IETF, August 1995.

[3] Ferrari, Domenico; Banerjea, Anindo; Zhang, Hui. *Network Support for Multimedia - A Discussion of the Tenet Approach*, Computer Networks and ISDN Systems, vol. 26, pp. 1267-1280, 1994.

[4] The ATM Forum, Technical Committee. *ATM User-Network Interface (UNI) Signalling Specification, Version 4.0*, `ftp://ftp.atmforum.com/pub/approved-specs/af-sig-0061.000.ps`, The ATM Forum, July 1996.

[5] The ATM Forum, Technical Committee. *ATM Forum Traffic Management Specification, Version 4.0*, `ftp://ftp.atmforum.com/pub/approved-specs/af-tm-0056.000.ps`, April 1996.

[6] RFC2211; Wroclawski, John. *Specification of the Controlled-Load Network Element Service*, IETF, September 1997.

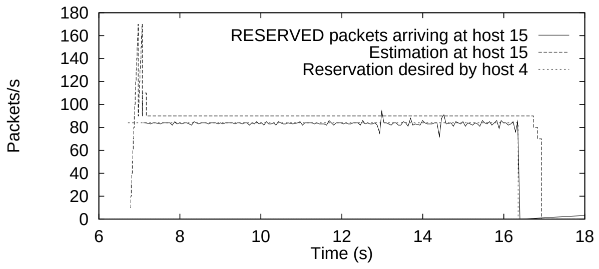

Figure 15: End-to-end reservation from host **4** to host **15**.

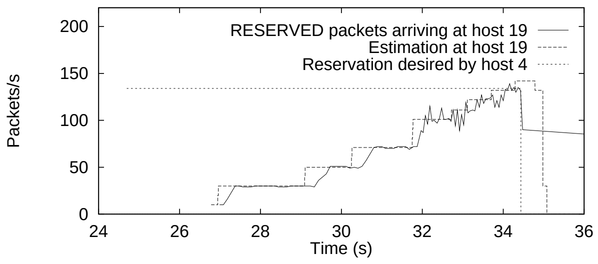

Figure 16: End-to-end reservation from host **4** to host **19**.

[7] Droz, Patrick. *Traffic Estimation and Resource Allocation Based on Periodical Wavelet Analyses*, `http://www.zurich.ibm.com/~dro/dsp96.ps`, September 1996.

[8] Gauthier, Eric; Giordano, Silvia; Le Boudec, Jean-Yves. *Reduce Connection Awareness*, `http://lrcwww.epfl.ch/scone/scone_paper2.ps`, Technical Report 95/145, DI-EPFL, September 1995.

[9] Diot, Christophe; Huitema, Christian; Turletti, Thierry. *Multimedia Applications should be Adaptive*, `ftp://www.inria.fr/rodeo/diot/nca-hpcs.ps.gz`, HPCS'95 Workshop, August 1995.

[10] Bolot, Jean; Turletti, Thierry. *Adaptive Error Control For Packet Video in the Internet*, Proceedings of IEEE ICIP '96, pp. 232–239, September 1996.

[11] Floyd, Sally; Jacobson, Van. *Random Early Detection Gateways for Congestion Avoidance*, `ftp://ftp.ee.lbl.gov/papers/early.ps.gz`, IEEE/ACM Transactions on Networking, August 1993.

[12] Differential Service for the Internet, `http://diffserv.lcs.mit.edu/`.

[13] Floyd, Sally; Jacobson, Van. *Link-sharing and Resource Management Models for Packet Networks*, IEEE/ACM Transactions on Networking, Vol. 3 No. 4, pp. 365-386, August 1995.

[14] Liebeherr, Jörg; Akyildiz, Ian F.; Sarkar, Debapriya. *Bandwidth Regulation of Real-Time Traffic Classes in Internetworks*, Computer Networks and ISDN Systems, Vol. 28, No. 6, April 1996, pp. 855 - 872.

[15] Le Boudec, Jean-Yves. *Network calculus made easy*, `http://lrcwww.epfl.ch/PS_files/d4paper.ps`, Technical Report 96/218, EPFL-DI, submitted to IEEE TIT, December 1996.

[16] Floyd, Sally. *Comments on Measurement-based Admissions Control for Controlled-Load Services*, `ftp://ftp.ee.lbl.gov/papers/admit.ps.Z`, July 1996.

[17] RFC1889: Schulzrinne, Henning; Casner, Stephen L.; Frederick, Ron; Jacobson, Van. *RTP: A Transport Protocol for Real-Time Applications*, IETF, January 1996.

[18] Floyd, Sally; Fall, Kevin. *Router Mechanisms to Support End-to-End Congestion Control*, `ftp://ftp.ee.lbl.gov/papers/collapse.ps`, Technical report, LBL, February 1997.

[19] RFC1633; Braden, Bob; Clark, David; Shenker, Scott. *Integrated Services in the Internet Architecture: an Overview.*, IETF, June 1994.

[20] RFC2201; Ballardie, Tony. *Core Based Trees (CBT) Multicast Routing Architecture*, IETF, September 1997.

[21] Maufer, Tom; Semeria, Chuck. *Introduction to IP Multicast Routing* (work in progress), Internet Draft `draft-ietf-mboned-intro-multicast-03.txt`, July 1997.

[22] McCanne, Steven; Jacobson, Van; Vetterli, Martin. *Receiver-driven Layered Multicast*, ACM SIGCOMM '96, August 1996.