Scalable Resource Reservation for the Internet

Werner Almesberger* <almesber@lrc.di.epfl.ch>,
Jean-Yves Le Boudec <leboudec@lrc.di.epfl.ch>,
Tiziana Ferrari <ferrari@lrc.di.epfl.ch>
EPFL DI-LRC, CH-1015 Lausanne, Switzerland

June 15, 1997

Keywords: Resource reservation, Internet, Scalability

Abstract

Current resource reservation architectures for multimedia networks don't scale well for a large number of flows. We propose a new architecture that aggregates flows on each link in the network. Therefore, the network has no knowledge of individual flows, and resource management functions traditionally implemented in the network (such as flow acceptance control) are delegated to hosts.

1 Introduction

Resource reservation architectures and protocols that have been proposed for integrated service networks (RSVP [1], ST-2 [2], ATM [3, 4]) borrow heavily from the architecture of the telephony network:

- routers or switches between the communicating hosts are required to store per-flow state information
- reservations, once granted, are stringent and conformance of traffic is carefully controlled (policing)
- most of the difficult work (e.g. admission control) is entirely handled by the network

*Contact author, phone +41 21 693 6621, fax +41 21 693 6610

The general intention is to provide a network service that is as deterministic as possible. While a highly deterministic service is certainly attractive, the complexity and lack of scalability of the aforementioned architectures and protocols makes their usefulness for many applications questionable. Particularly Internet telephony creates a need for very inexpensive means to obtain a dependable quality of service for a very large number of concurrent flows.

This goal can be achieved by aggregating flows so that the reservation mechanism only needs to be aware of a comparably small number of (aggregated) flows. The architecture we propose in this paper goes even beyond concepts for aggregation on top of traditional reservation architectures (e.g. [5] for ATM) in that it makes aggregation the standard behaviour 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 for Internet telephony) 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 independently by sources and routers, using a "learn by example" procedure. The accepted rate is guaranteed as long as there is a minimum activity by the source. The reservation disappears by timeout after the source has stayed idle for a while. Figure 1 shows an idealized view of our model, and a more detailed example is given in section 2.5. 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.

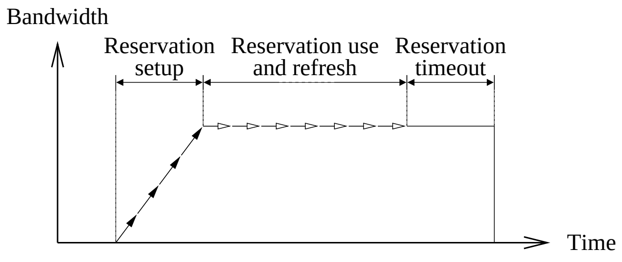


Figure 1: Idealized reservation procedure.

A key feature of 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 rates;
- routers maintain reservations by learning, namely, by monitoring the actual reserved traffic.

We discuss these two design directions in the rest of this section. Section 2 describes the fundamental architecture. Section 3 elaborates on that and also points out areas where more research is needed. Finally, protocol operation is illustrated with some simulation results in section 4 and the paper concludes with section 5.

1.1 The TCP case

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. Responsibility for congestion control is given entirely to the end systems (e.g. TCP), 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.

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 needed, policing functions can be implemented per flow at some network access points; we believe that such policing is not needed at inter-carrier exchanges, or in general anywhere beyond Internet access points.

Because flow acceptance control is inherently flow-specific, delegating it to end systems is a requirement for enabling routers to efficiently aggregate arbitrary flows.

The proposed architecture slightly extends the traditional Internet design by introducing the concept of packet types to distinguish reserved traffic from best-effort traffic. This also allows routers to give more precise admission control indications than just a simple forward–or–discard decision.

1.2 Adaptive applications

The desire to run multimedia applications over the current best-effort Internet with all its imperfections has motivated the development of increasingly sophisticated adaptive applications.[6, 7] Adaptive applications tolerate variations in packet loss rates, in bandwidth, and in delay.

Of course, even adaptive applications have certain minimum requirements. This is typically a minimum bandwidth, below which no useful operation is possible. If additional bandwidth is available, it is used to improve the service (e.g. better audio quality).

The proposed architecture aims mainly to ensure that adaptive applications can obtain their minimum bandwidth. The service goals are similar to the ones of the INTSERV controlled-load service [8]: Availability of enough bandwidth is guaranteed but all other parameters (such as delay, loss rate, etc.) remain unspecified, although an application can assume that they are in a "reasonable" range.

The presence of adaptive applications also implies that there is usually enough best-effort traffic in the network that only a small fraction of the total bandwidth will be used for reserved traffic and that resource shortage for reserved traffic will be an infrequent situation.

1.3 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 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 an estimation module used by sources, routers, and destinations. Assuming that sources emit traffic in regular periodic patterns, a simple implementation could just count the number of requests during a time interval and use that to predict the increase of total reserved bandwidth. Note that the period of a source must be reasonably short compared to the observation interval for such measurements to be meaningful.

The same principle is also used to detect decreases in the use of reserved bandwidth: Routers monitor the amount of reserved traffic and adjust reservations automatically if sources reduce their bandwidth or stop sending. We describe this simple implementation in sections 2.3 and 2.5.

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.

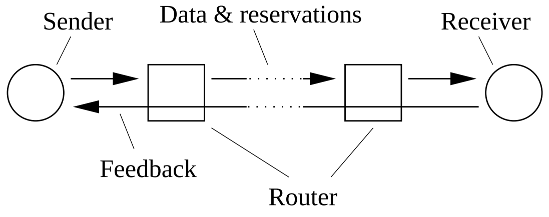


Figure 2: Network structure overview.

Figure 2 illustrates the operation of the two protocols:

- Data packets with reservation information are sent from the sender to the receiver. The reservation information is processed by routers. They may modify the reservation information or they may also discard packets.
- The receiver sends feedback information back to the sender. Routers only forward this information; they don't need to process it.

Routers monitor the effectively present reserved traffic and adjust their reservations accordingly.

2.1 Reservation protocol

The reservation protocol is used in the direction from the sender to the receiver. It is processed by the sender, the receiver, and the routers between them. In order to simplify processing of the reservation protocol, the reservation information is represented as a *packet type* which is included in normal data packets.[1]

The reservation protocol uses three packet types:

Reserved The reservation has already been established (and confirmed). The packet of type *reserved* uses that reservation.

Request A reservation is needed for packets like the current one, but a reservation has not yet been confirmed (e.g. because no request was sent yet or because the feedback hasn't reached the sender yet).

Best effort No reservation is needed.

Packet types are initially assigned by the sender, as shown in figure 3. 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 covers the current packet. If yes, the packet is sent as *reserved.* Otherwise, an additional reservation is requested by sending the packet with the *request* flag.

Each router performs the following processing (see also figure 4):

[1]The encoding is yet unspecified. One possible approach is to define a new IP option to carry the packet type.

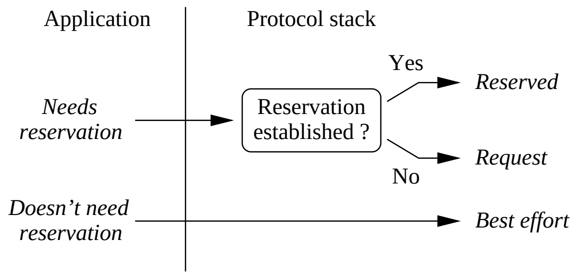


Figure 3: Initial packet type assignment by sender.

- If a *request* can be accepted, the reservation is made and the packet is forwarded unchanged. Otherwise, its type is set to *best effort* and best-effort processing is performed.

- If a *reserved* packet is received, the router verifies that a suitable reservation exists. This is normally the case and the packet is forwarded unchanged and with priority over best-effort traffic. If no reservation exists, either a protocol error or a route change has occurred (see section 3.3). In order to stabilize the reservation, the packet type is changed to *request* and request processing is performed.[2]

Furthermore, *best-effort* packets may be discarded during congestion.

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.

[2]Considering that *reserved* packets will "magically" become requests if necessary, one may be tempted at this point to avoid the use of a *request* packet type entirely. At least in the given framework, this does not work. Requests are needed to isolate already established reservations from increases or new reservations: if packets from "old" and "new" reservations were both of type *reserved*, a router experiencing resource shortage had no way of knowing which ones to degrade or even to discard and it would consequently also penalize the "old" reservations.

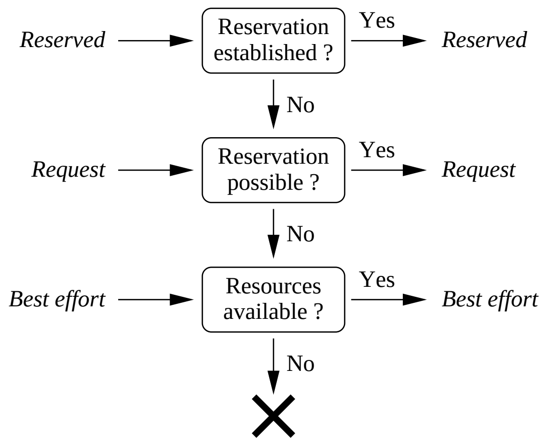


Figure 4: Packet type processing by routers.

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 by executing an algorithm like the one routers use to estimate the actual resource use. 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 requests and is used by the function that decides if packets are sent as *reserved* or as *request*.

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 [9].

2.3 Reservation dynamics

Reservation are set up for the traffic profile reflected by the requests sent by the source. A router can for instance count 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. The use of reservations can be simply measured by counting the number of *reserved* packets that are received in a certain interval.

With such measurements for time t, the amount of resources to reserve at time $t+1$ can be predicted as follows:

$$reserve_{t+1} = requests_t + rsv\_seen_t$$

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. With a hold time of h observation intervals, the reservation is computed as follows:

$$\begin{aligned} reserve_{t+1} = & \max(requests_{t-h+1} + \\ & rsv\_seen_{t-h+1}, \ldots, \\ & requests_t + rsv\_seen_t) \end{aligned} \quad (1)$$

The definition and evaluation of the algorithms for reservation calculation in hosts and routers is still ongoing work. The algorithms above should serve only as examples.

We call this algorithm an *estimator*, since it attempts to estimate, based on past traffic, the resources that will need to be reserved in the future. Figure 5 shows how the estimator algorithm is used in all network elements:

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

- Routers use the estimator for packet-wise admission control and also to detect anomalies (see section 3.3).

- 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.

A source always uses the minimum of the (optimistic) estimation of the reservation at the next router and the (conservative) feedback.

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 observation interval. Routers can detect this by the lack of *reserved* packets and they consequently refrain from increasing the reservation. The feedback that is returned to the sender may also show an increased number of requests.[3] The sender must not interpret those requests as a direct increase of the reservation, because the routers didn't either. Instead, the sender uses the same algorithm as the routers to correct the feedback information accordingly.

2.4 Resource reservation in a router

This section gives an example of how resource reservation can be handled in simple a 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 examined and acted upon (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.

[3]Some of them can, however, also be refused in the network and either become best-effort or even get discarded.

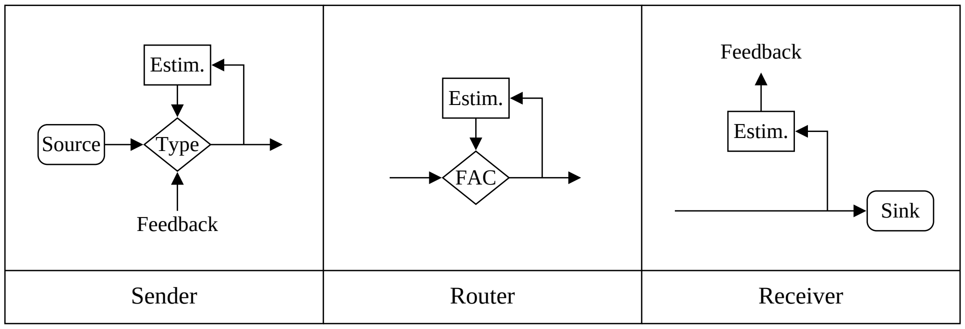


Figure 5: Algorithms in the senders, routers, and receivers.

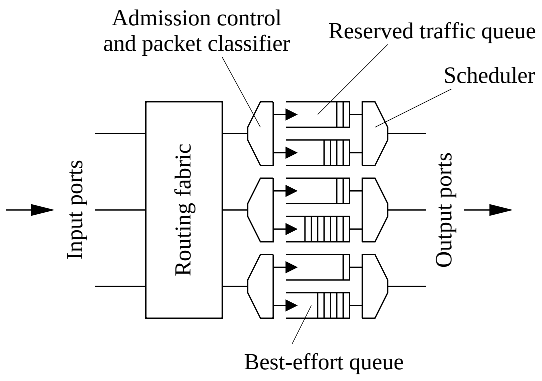
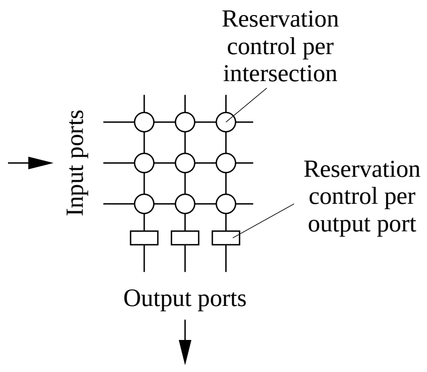


Figure 6: Example router.



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

Figure 7: Reservation control in router.

Placing the reservation mechanisms 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).

2.5 Reservation example

In this section, we illustrate the operation of the reservation mechanism in a very simple example. The network we use is shown in figure 8: 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.

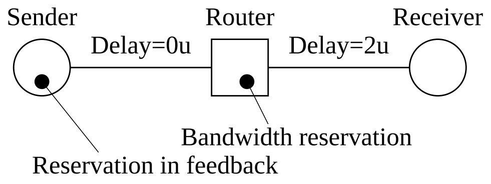


Figure 8: Example network configuration.

The bandwidth reservation in the router and the reservation that has been acknowledged in a feedback message from the receiver are measured. In figure 9, 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 router 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 router discovers that it has established a reservation for six packets in four time units, but that the source has only sent four packets. If therefore 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 *request*s. At point (3), the next observation interval ends at the router 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 no change after that.

3 Additional aspects

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

3.1 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 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.

3.2 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 [10]).

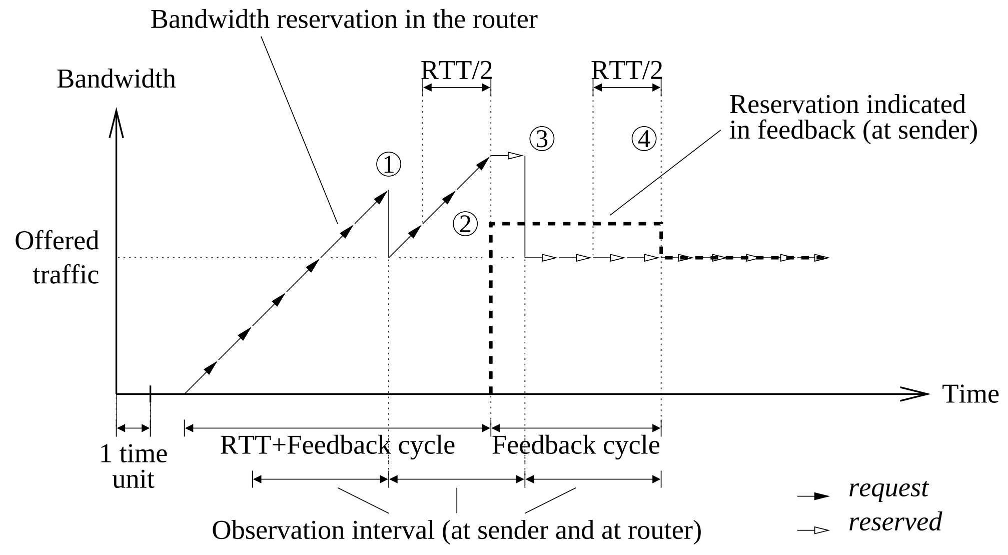
If the network implements a packet type for inelastic best-effort traffic[4] or generally a lower priority type than normal best-effort traffic, that type should be used when degrading *request* packets. Otherwise, a more aggressive discard policy has to be used for those packets. This could for instance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

3.3 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” [11]) 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**). In order to limit the impact of this, the first router that detects the change should change *reserved* packets to *request*. In figure 10, routers **A** and **B** can orderly try to es-

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



Bandwidth reservation in the router
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 and at router)
request
reserved

Figure 9: Protocol operation example.

tablish reservations on links **a** and **b** (and on all downstream links) if router **A** changes the type of redirected packets. 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 to requests.

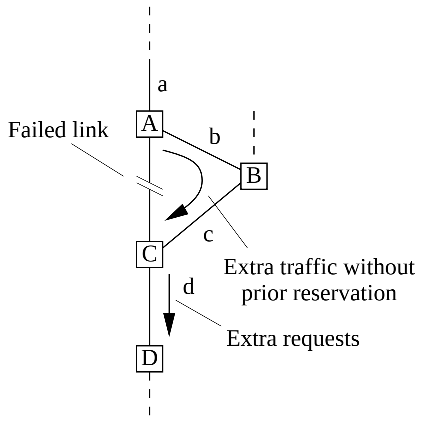


Figure 10: Route change example.

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 2.3 and the same mechanisms can be used to overcome it.

3.4 Discussion of the estimation module

We have presented in section 2.3 an estimation module based on counting the number of bytes in request and reservation packets per time interval. We discuss now some implications of this estimation module. The estimated bandwidth required for one output for the tth time interval is $reserve_t$. Call T the length of the time interval. If the estimation is correct, this means that an arrival curve for the aggregate flow is $\alpha(u) = (T + u)reserve_t$. If the aggregate flow is served at a line rate c, this implies that the maximum delay variation for this flow is $T\frac{reserve_t}{c}$ [12]. With this estimation module, the value of T is thus linked to the delay imposed on reserved traffic.

It is possible to modify the estimation module as follows in order to remove this dependency. In equation 1, define $requests_t$ as the *deterministic effective bandwidth* for a delay bound D [12] of the flow of requests over the tth time interval, and rsv_seen_t the deterministic effective bandwidth for the same delay bound D of the flow of reservations. The deterministic effective bandwidth is the amount of bandwidth that is required to serve an observed flow within a given delay bound. This modification makes the estimation module more complex, but it makes it possible to have an observation interval T much larger than the delay bound D. A more detailed analysis of the estimation module is the object of current research.

3.5 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 [13, 14], 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.

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 *reserved* 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 11.

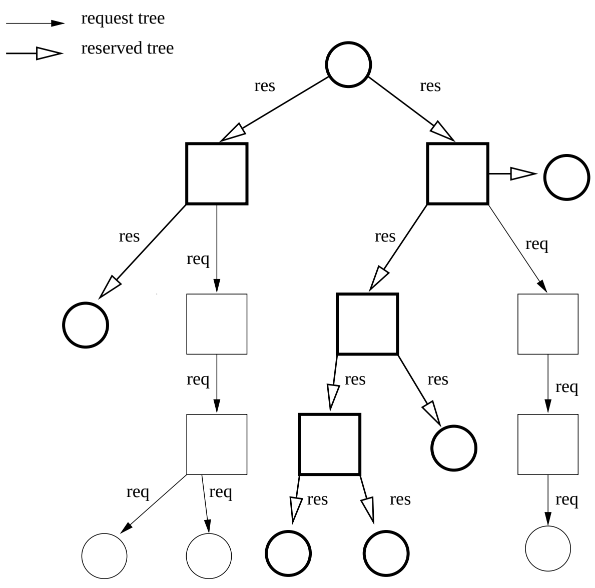


Figure 11: Request and reserved multicast group.

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,

[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, nonconforming 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.

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 ((pck_addr is multicast) and
        (pck_type == rsvd)) {
  forward pck to reserved group;
  if (router is in the request group) {
        newpck = copy(pck);
        newpck_type = req;
        forward newpck to request group;
  }
}
```

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 [15] could be used.

Hosts which can apply for the same reservation level, are associated 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.

4 Simulation

The graphs in this section show the simulated performance of the proposed architecture. The configuration is shown in figure 12.

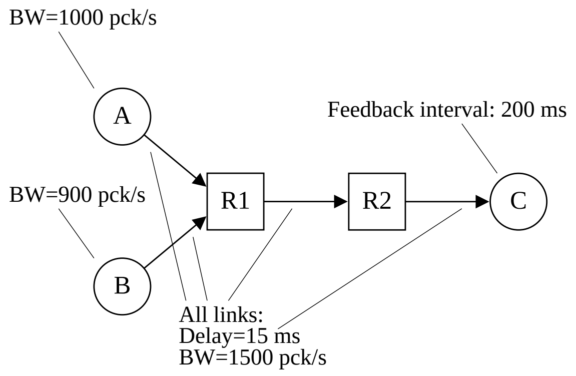


Figure 12: Simulation network configuration.

Senders **A** and **B** send via the routers **R1** and **R2** to the receiver **C**. Each link in the network can handle 1500 packets per second and has a delay of 15 ms. **A** sends 1000 packets per second, **B** send 900, and the receiver returns feedback every 200 ms.

Figure 13 shows the reservation setup when only **A** is sending: After 290 ms (the round-trip-time of $2 \cdot 3 \cdot 15$ ms plus 200 ms until the first feedback is sent), the feedback arrives and **A** can start sending reserved traffic. The reservations in the network stabilize and stay constant shortly thereafter.

Figure 14 shows reserved traffic sent by **A** and **B** and the (aggregated) reservation set up at **R1** when

[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.

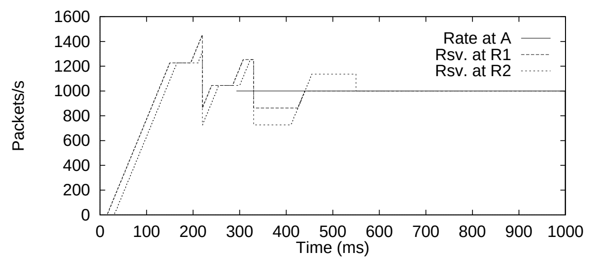


Figure 13: Reservation setup with only **A** sending.

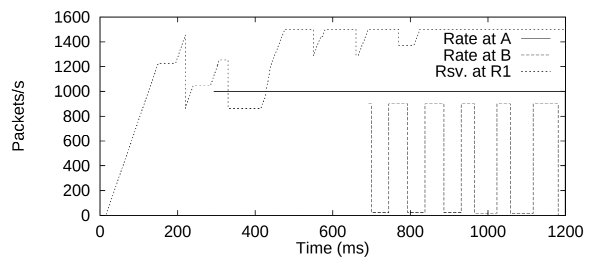


Figure 14: Reservation setup with **A** and **B** sending.

also **B** is sending. Since the sources try to send at 900 plus 1000 packets per second on a path that can only accept 1500 packets per second, only the first sender obtains its full reservation. It is clearly visible that **B**, who starts sending at 400 ms, only obtains slightly more than half of its reservation and therefore has to send much of its traffic as best-effort. **A**'s sending rate is not affected by all this.

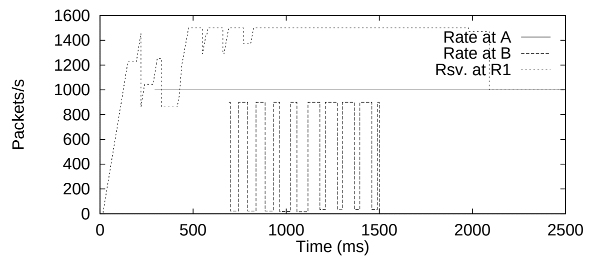


Figure 15: Reservation timeout at **B**.

Finally, figure 15 shows how the reservation times out in the network when **B** gives up and stops sending at 1500 ms.

5 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 basic implementations of core elements of the architecture, and showed some illustrative simulation results. Further research is needed to resolve open issues needed for a comprehensive specification and to improve efficiency, robustness, and versatility of the algorithms and procedures outlined in this paper.

References

[1] Braden, Bob; Zhang, Lixia; Berson, Steve; Herzog, Shai; Jamin, Sugih. *Resource ReSerVation Protocol (RSVP) – Version 1 Functional Specification* (work in progress), Internet Draft `draft-ietf-rsvp-spec-15.ps`, May 1997.

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

[3] 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.

[4] 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.

[5] 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.

[6] 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.

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

[8] Wroclawski, John. *Specification of the Controlled-Load Network Element Service* (work in progress), Internet Draft `draft-ietf-intserv-ctrl-load-svc-05.txt`, May 1997.

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

[10] 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.

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

[12] 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.

[13] Ballardie, Tony. *Core Based Trees (CBT) Multicast Routing Architecture* (work in progress), Internet Draft `draft-ietf-idmr-cbt-arch-06.txt`, May 1997.

[14] Maufer, Tom; Semeria, Chuck. *Introduction to IP Multicast Routing* (work in progress), Internet Draft `draft-ietf-mboned-intro-multicast-02.txt`, March 1997.

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