

Charging and Accounting Protocol for IP Multicast over ATM with QoS Support

by

Nguyen Tuong Long Le

Submitted to the Department of Telecommunications Engineering
at
the Technical University of Berlin

in Partial Fulfillment of the Requirements for the
Pre-diploma thesis

September 1998

Acknowledgements

First and foremost, I thank my family and my friends for their support, encouragement, and understanding. I wish to thank Prof. Dr. Adam Wolisz and Dr. Micheal Smirnov for giving me the opportunity to do this thesis. My special thanks are due to my supervisors, Dr. Georg Carle and Tanja Zseby, who have spent very many hours discussing with me, guiding me to conduct research, and proof-reading this thesis. I am very grateful to Henning Sanneck and Davinder Pal Singh for their support in RSVP protocol that has greatly helped me during the time I implemented the code of the charging and accounting protocol into the existing code of RSVP.

Contents

Acknowledgements	2
List of figures	5
Abstract	6
1. Introduction	7
1.1 Motivation and Background	7
1.2 Overview	7
2. Service Scenarios and Related Work	9
2.1 Service Scenarios	9
2.1.1 Video Conferencing and Distributed Simulation (War Gaming)	9
2.1.2 Lecture Broadcasting	9
2.1.3 Combination	9
2.2 Related Work	11
2.2.1 IP Multicast and IGMP	11
2.2.2 MARS and IP Multicast over ATM	12
2.2.3 Resource ReSerVation Protocol	13
2.2.4 EARTH and MIS	14
3. General Issues of Charging and Accounting in Computer Networks	17
3.1 A Simple and General Formula	17
3.2 Charging and Accounting in ATM Networks	18
3.2.1 Charging and Accounting for Point to Point Connections	19
3.2.2 Charging and Accounting for Point to Multipoint Connections	19
3.3 Charging and Accounting in Conjunction with RSVP and IP Multicast	21
4. Protocol Requirements	23
5. Architecture of the Charging and Accounting Protocol	25
5.1 Terminology	25
5.2 Service Scenarios and Potential Solutions	25
5.2.1 Service Scenario 1: one ATM Network in Combination with MICAS	26
5.2.1.1 Sender-paying Model	26
5.2.1.1.1 MICAS is in Charge of Computing the Cost of the Multicast Tree	26
5.2.1.1.2 Ingress Node is in Charge of Computing the Cost of the Multicast Tree	26
5.2.1.2 Receiver-paying Model	27
5.2.1.2.1 MICAS is in Charge of Computing the Cost of the Multicast Tree	28
5.2.1.2.2 Ingress Node is in Charge of Computing the Cost of the Multicast Tree	28
5.2.2 Service scenario 2: Multiple Legacy LANs Attached to an ATM Network	28
5.2.2.1 Sender-paying Model	29
5.2.2.2 Receiver-paying Model	29
5.3 The General Charging and Accounting Protocol	30
5.3.1 Design issues	30
5.3.2 Protocol specification	33
6. Summary and Future Work	42

6.1 Summary	42
6.2 Future work	42
Appendix A RSVP-CAP Interface	43
A.1 The RSVP-LPM Interface	43
A.2 The RSVP-CAP Interface	44
Appendix B Implementation Details	48
Appendix C Testing of the Implementation	52
C.1 Testing of the Implementation in the simulated RSVP environment	52
C.2 Testing of the Implementation in the real RSVP environment	64
Appendix D References	68

List of Figures

Figure 1: Disjoint multicast trees	10
Figure 2: Partly joint multicast trees	10
Figure 3: IP multicast and IGMP	12
Figure 4: Message flow of RSVP in MIS architecture	16
Figure 5: Sharing the cost of a point to multipoint connection in ATM networks among the receivers	20
Figure 6: Sharing the cost of a link of receivers having different QoS levels	23
Figure 7: RSVP-CAP interface	33
Figure 8: MSC for CAP in receiver-paying model	37
Figure 9: Exchange of charging and accounting information in receiver-paying model	38
Figure 10: MSC for CAP in sender-paying model	40
Figure 11: Exchange of charging and accounting information in sender-paying model	41
Figure 12: Functionality and data structure of a CAP module	50
Figure 13: Data structure of a CAP module containing charging and accounting information	51
Figure 14: State transition diagram of a CAP module in receiver-paying model	52
Figure 15: Message exchange and functionality of a CAP module located inside a router	53

Abstract

This pre-diploma thesis addresses the issues of charging and accounting for the current Internet. We first take a look at different pricing structures, service scenarios and paying models for the Internet. We then introduce our charging and accounting protocol for IP multicast over ATM with QoS support. Our charging and accounting protocol (CAP) is an extension of the RSVP set-up protocol with the capability of keeping track of technical costs incurred by the users and sharing these costs fairly among them. Finally, we present the implementation details and the interface between RSVP and our charging and accounting protocol. Although our protocol is originally designed to support the Multicast Integration Server [Smir97], [CoSS98], [CaCS98] with the capability of charging and accounting, we extend it to run in conjunction with RSVP or a similar signaling protocol on any network technologies.

1. Introduction

1.1 Motivation and Background

In recent years, the phenomenal growth of the Internet has caused it to make an amazingly fast transition from a cooperative research network to a competitive and commercial network. Besides the Internet's "classical" services such as ftp, email etc. with a single quality of service called "best effort"¹, many companies and research institutions have been making great efforts to deploy new applications such as video on demand, video conferencing, telephony on the Internet etc.

Unlike the "classical" services, many of these new applications have more stringent requirements such as high bandwidth, low delay, and minimum jitters and will be sharply degraded when the network does not provide them with an adequate quality of service. This kind of requirements was not taken into account when the Internet was designed and implemented thirty years ago. In order to fulfill these requirements, network resources must be reserved along the data path as long as they are still scarce.

Much research has been done in this area and many problems are expected to be solved in the near future. However, in a non-cooperative environment many users would require the best service categories if there were no mechanisms for charging. A reservation protocol would be contra-productive without a mechanism for charging in a non-cooperative environment. Thus, a charging protocol is needed here to give the users incentives for reasonable behavior and efficient usage.

Moreover, many engineers and economists believe that the current Internet's congestion problems are due to its ineffective pricing structure, namely flat rate pricing, where prices depend only on the rate of the access pipe which connects a customer to his provider [Fird97]. This pricing structure does not give users incentives to behave well and keep them from being too "greedy". A good pricing structure, e.g. usage-based pricing structure, can effectively help to alleviate the current congestion problems of the Internet.

The original design of the Internet based on the philosophy of "best-effort" QoS and its historical dependency on government subsidies resulted in a meager research interest in the area of charging and accounting for the Internet. However, the privatization and transformation of the Internet have led into a situation, in which network service providers are badly in need for a solution of charging, accounting, and controlling network resources.

All these reasons have recently resulted in a suddenly strong interest in the field charging and accounting for the Internet.

1.2 Overview

Although the topic of charging and accounting in computer networks has been recently gaining much attention and some research has been done in this area for a few years, there is

¹ "Best effort" means the network does not guarantee whether data packets arrive at the receiver or corrupted or delayed. The network simply tries to do its best to transmit data packets. Data packets might be lost, corrupted, duplicated or reordered.

no consensus for a standard solution until now. The most debates center on the paying model (sender-paying model vs. receiver-paying model) and the pricing model (flat pricing vs. usage-based pricing [Shen96]).

We claim that the flat pricing model is not appropriate for new kinds of applications with high requirements for network resources because it does not reflect the network resources an application consumes or reserves. Thus, the flat pricing model does not give users incentives for efficient usage of network resources and good behavior. Therefore, the flat pricing model cannot help solve the current congestion problems of the Internet.

In our viewpoint, the paying model (sender-paying model vs. receiver-paying model) depends upon the type of applications. In this work, we try to address both models. However, the receiver-paying model is more attractive [CaSZ98] and reasonable in the case of IP multicast services with or without QoS support. This is because the receiver-paying model is fully in line with the service model of IP multicast and RSVP (sender simply sends its packets and is not aware of individual receivers and the QoS they receive while receivers have to take care of how to join a multicast group and how to reserve QoS).

Our document is organized as follows:

In chapter 2, we present service scenarios that are considered to be typical for applications of IP multicast and a short introduction to some research topics that are related to this work.

In chapter 3, we discuss the general issues of charging and accounting in computer networks and the special issues of this topic in ATM and IP networks.

In chapter 4, we list the requirements we consider to be important and which should be fulfilled by a charging and accounting protocol.

We then proceed to chapter 5 where we discuss the design alternatives of a charging and accounting protocol in the MIS architecture. We finally conclude this chapter by proposing our general charging and accounting protocol.

Chapter 6 sums up our work with a summary and an outlook of future research.

The appendices give the interested readers more detailed information about our prototype implementation.

2. Service Scenarios and Related Work

In section 2.1, we will present to the readers some service scenarios that are considered to be typical for the sender-paying and receiver-paying model. These service scenarios provide an evidence that there is a need of such protocols as IP multicast and charging and accounting protocols in order to share the costs among the users. In section 2.2, we provide the readers with an introduction to some related topics that should be familiar to someone wishing to understand our charging and accounting protocol.

2.1 Service Scenarios

Many kinds of new applications are based on the service model of IP multicast with QoS support. Thus, a sane pricing model should reflect the costs of the network resources consumed or reserved by the users and share these costs fairly among them. In our point of view, all kinds of new applications can be basically divided into two categories: those based on the sender-paying and those based on receiver-paying model. Following, we present some service scenarios that are typical for these two categories of pricing model.

2.1.1 Video Conferencing and Distributed Simulation (War Gaming)

Video conferencing and distributed simulation (war gaming) are considered to be typical for the sender-paying model. In this service scenario, many users participate in a session and each user pays the costs incurred by himself when he sends data to other participants. In case that the paths along the multicast tree¹ taken by each participant's data are disjoint, the problem is reduced to how to determine and compute the costs incurred by each participants. However, the problem is much more complex when the paths are shared by several multicast trees. In this case, the costs for the common paths must be shared in a fair way among the senders.

2.1.2 Lecture Broadcasting

Lecture broadcasting is a typical example for the receiver-paying model. In this service scenario, there is only one sender² at a time that sends its data to several receivers. The problem here is how to find a method to fairly share the costs of the multicast tree among the receivers. The problem is more complicated when the receivers have different QoS. In section 3.3, we will present a solution proposed by Shai Herzog to tackle this problem.

2.1.3 Combination

There are some service scenarios that are a combination of the two service scenarios above. Following, we name a few service scenarios for which a solution can be found by combining the solutions of the problems in the two first service scenarios:

- A single sender and several receivers share the costs of the multicast tree.

¹ We will learn more about IP multicast and multicast tree in section 2.2. At the moment, it is sufficient to know that IP multicast is an efficient technique to send data from a single sender to several receivers and the multicast tree is the path taken by the data of the sender.

² However, the sender might not be fixed in some cases (e.g., a lecturer can take the role of a receiver while a student is asking questions.)

- A single sender sponsors the costs on some certain surrounding networks. Receivers outside of these networks must pay the rest of the costs if they wish to receive multicast traffic from that sender.

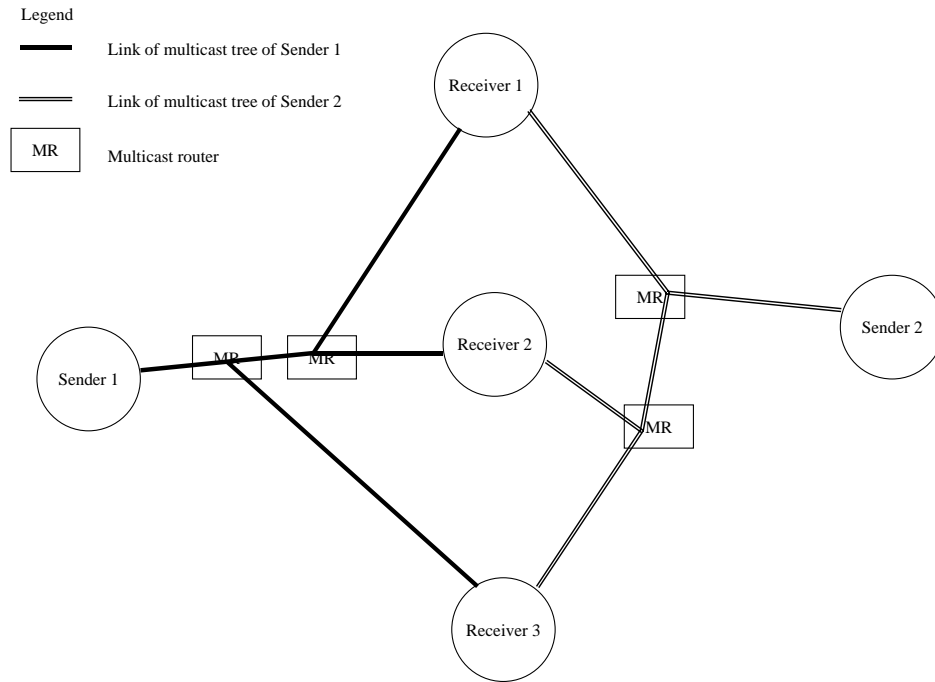


Figure 1: Disjoint multicast trees

Figure 1 shows a service scenario with two senders and three receivers. In this scenario, the two multicast tree are disjoint.

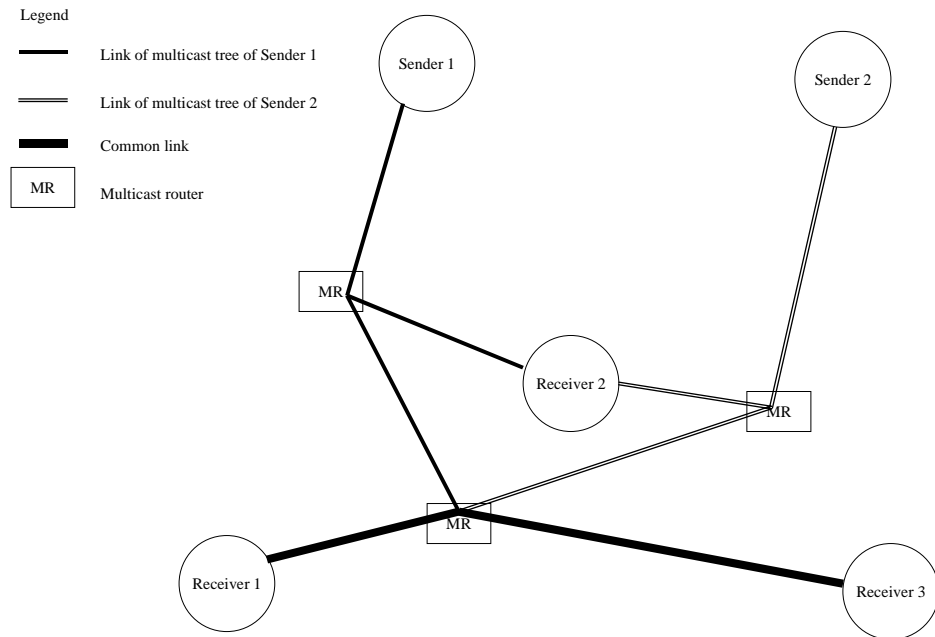


Figure 2: Partly joint multicast trees

Figure 2 shows a service scenario with two senders and three receivers. In this scenario, the two multicast trees have the links connected to receiver 1 and receiver 3 in common. A solution for this service scenario is beyond the scope of this work.

2.2 Related Work

In this section, we provide an introduction to some topics that are related to our work. Note that we focus only on the related topics to our work. Readers who do not have an adequate background in computer networks may refer to some of the following textbooks on data communications and computer networks: [Come95], [Hals95], [Spra91], [Stev93], [Tane96], and [Thom96].

We start this section with an introduction to IP multicast. IP multicast provides an efficient technique in several service scenarios, such as video conferencing and lecture broadcasting when a sender sends its data to multiple receivers. We then deliver a short introduction to a current research topic, IP multicast over ATM. We will also present a short introduction to RSVP, a signaling protocol that is used to reserve network resources in order to guarantee qualities of service. As we will see later, RSVP is used as a transport protocol to communicate the charging and accounting information between the CAP modules. Next, we make the readers familiar with the problems of IP multicast over ATM and their current solutions. We complete this chapter with an overview of the Multicast Integration Server (MIS) architecture. The MIS architecture provides an efficient solution for IP multicast over ATM with QoS guarantee.

2.2.1 IP Multicast and IGMP [Deer89]

IP multicast means that the network delivers the data traffic from a sender to a set of receivers that forms a multicast group. IP multicast helps to conserve bandwidth because only one packet must be sent by the sender that is replicated only at branching points of the multicast distribution tree. In IP multicast, the sender simply sends its packets to the multicast address as in the case of IP unicasting and does not care about the delivery. If the receivers want to receive the multicast traffic, they must do something about it. This model is called receiver-oriented, contrasting with the sender-oriented model of the ATM technology which we will see in section 2.2.2. When some receivers of a multicast group are not on the same local network with the sender, special routers are needed to forward the multicast traffic to these receivers. These special routers are called multicast routers. Hosts participating in multicast groups and multicast routers use the Internet Group Management Protocol (IGMP) to communicate with each other.

IGMP has been invented by Steve Deering [Deer89]. In IGMP version 1, multicast routers poll the hosts to refresh their knowledge of memberships of the multicast groups by periodically broadcasting IGMP Queries on the local networks. If no IGMP Reports for a multicast group are received after some number of IGMP Queries, the multicast routers assume that all the receivers on the local network have already left the multicast group and they do not need to forward remotely-originated multicast traffic for that group onto the local network anymore.

Hosts respond to IGMP Queries by sending IGMP Reports, reporting their membership to multicast groups. However, they do not immediately send the IGMP Reports after receiving an IGMP message but start a random delay timer for each of their IGMP Reports. An IGMP

Report for a multicast group is sent by a host when the timer for it has expired and when no IGMP Reports for that multicast group have been heard on the local network. If a host hears an IGMP Report for a multicast group that the host is also about to send an IGMP for, it stops the timer for its IGMP Report and do not send an IGMP Report for that multicast group. This algorithm helps to avoid message implosion, i.e. multiple hosts send IGMP Reports for the same multicast group while only one IGMP Report is sufficient.

In conjunction with IGMP, the Distance Vector Multicast Routing Protocol (DVMRP) [Wait88] is a routing protocol that is used to forward multicast traffic to remote receivers that are not on the same local network with the sender. The majority of the Mbone¹ regions are currently interconnected by DVMRP [Mauf97]. Besides DVMRP, there are also other multicast routing protocols, like Multicast Extensions to OSPF (MOSPF) [Moy94] and Protocol-Independent Multicast (PIM) routing protocol(s) [Est97, Deer97]. [Mauf97] provides a short but very good introduction to IP multicast.

Figure 3 depicts the functionality of IP multicast and IGMP.

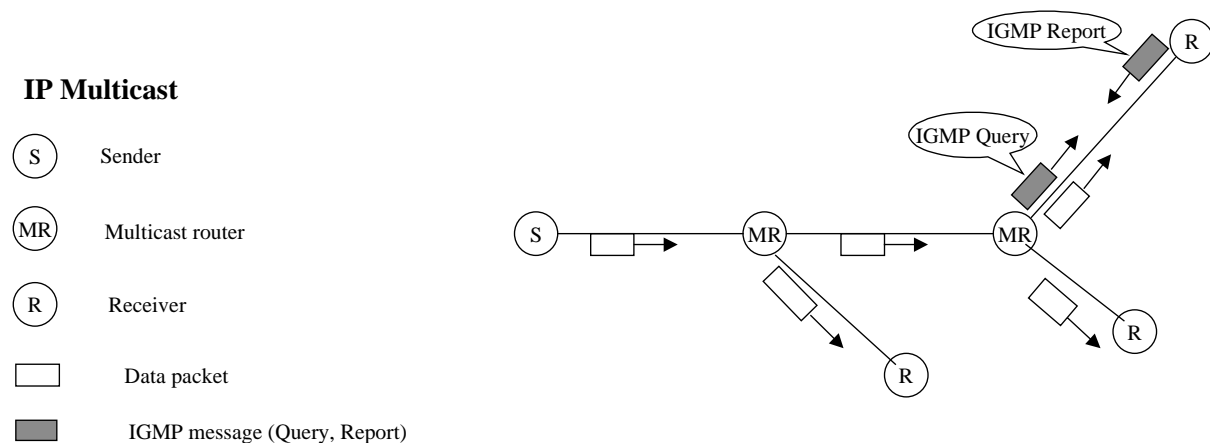


Figure 3: IP multicast and IGMP

2.2.2 MARS and IP Multicast over ATM [Armi96]

ATM is a high-speed technology that supports quality of service. ATM is based on cell technology and can hence offer a good multiplexing scheme that provides transmission services with low jitters. This beautiful feature makes ATM very suitable for isochronous and real-time applications, such as video conferencing and Internet telephony. ATM has gained much attention in recent years and has been considered as a very promising network technology. In recent years, much research has been done for improving the transmission technology of ATM and for providing IP services over ATM networks.

In contrast to IP being a connectionless packet-switching protocol, ATM is a connection-oriented technology. Although it is widely agreed that ATM's cell technology is the correct choice, there has been a continuous debate whether a high-speed technology like ATM should be connection-oriented or connectionless. The different philosophies of IP and ATM make it difficult to map IP's services to ATM services. Another difficulty providing IP multicast over

¹ The Mbone is a network that is overlaid on top of sections of the Internet. Its function is to support the multicast applications while the ubiquitous deployment of multicast-capable routers in the Internet is not yet completed.

ATM networks is that IP multicast is originally used on broadcast networks like Ethernet while ATM does not support broadcasting.

Multicast in ATM is supported by point to multipoint unidirectional virtual channels (VCs). In ATM, the sender must know the addresses of the intended receivers and explicitly establish a VC with itself as a root and the receivers as the leaf nodes. When a new receiver wants to receive data traffic from the sender, the sender must explicitly add this receiver as a new leaf node to the point to multipoint connection. When a receiver does not want to receive data traffic from the sender anymore, the sender must explicitly remove that receiver from the point to multipoint connection. Because the point to multipoint connection is unidirectional, the receivers have no way to use this connection to tell the sender about their membership. Thus, there must be an extra mechanism that enables the senders and receivers to communicate their membership. Moreover, IP and ATM have fully independent address space and there is no way to know the ATM address of a network interface given its IP address.

Armitage has proposed a Multicast Address Resolution Server¹ (MARS) as a solution for resolving an IP multicast address to a set of ATM addresses of the group members participating in a multicast group. It enables the receivers to communicate with the sender about their membership via a registry server. MARS acts as a registry server in a logical IP subnet, associating the IP multicast address with the ATM address of the group members. The sender queries the MARS when an IP multicast address need to be resolved to the set of ATM addresses of the endpoints of the VC. Endpoints trigger the MARS to update its information about their group membership when they need to join or leave particular IP multicast groups. To allow asynchronous notification of group membership changes the MARS manages a point to multipoint VC to all endpoints desiring the multicast support.

2.2.3 Resource ReSerVation Protocol [Brad97]

The Resource ReSerVation Protocol (RSVP) is a signaling protocol that is used to reserve network resources along the data path in order to provide users with quality of service (QoS). In RSVP, the sender does not care about reserving the network resources. It simply sends its unicast or multicast data. If the receivers wish to have better quality of service than the „best-effort“ service, they have to take care of reserving the network resources along the data path.

In RSVP, the sender periodically sends Path messages for each of its data flows. The Path messages contain information about the characteristics of the flows. The Path messages are sent to the multicast group address in case of IP multicast and to the unicast address of the receiver in the case of IP unicasting.

Based on the information contained in Path messages, the receivers decide what kind of reservations and qualities of service they would like to have. They create a Reservation message (Resv message) that reflects their desired QoS level and send it on a hop-by-hop basis upstream to the sending host. When a router receives the Resv message, the router examines it and tries to aggregate it with other reservations already made in place.

If there already exists a reservation with similar characteristics for a given sender, the router does not need to reserve network resources for the new receiver. Otherwise, the router has to

¹ MARS is the current proposed standard of the IETF (Internet Engineering Task Force) ION working group for IP Multicast over ATM.

reserve appropriate network resources for the receiver and forwards the reservation upstream. If the router fails to make the reservation required by the receiver, it sends a reservation error message (ResvErr message) downstream to the receiver notifying that it cannot satisfy the receiver's requirement. The ResvErr message is propagated hop-by-hop. At each hop, the IP destination address is the unicast address of the next downstream hop.

The Resv messages propagate upstream on a hop-by-hop basis towards the sender like described above and thus reserve the network resources along the data path in order to provide the data traffic of the sender with the wanted QoS. When the Resv messages finally arrive at the sender host, it can use the traffic parameters specified in the Resv messages to adjust its data traffic and satisfy the requests of the receivers if possible.

2.2.4 EARTH and MIS [Smir97, CoSS98]

EARTH: The EAsy IP Multicast Routing THrough ATM clouds (EARTH) protocol is another solution for resolving an IP multicast address to a set of ATM addresses. In comparison to MARS, EARTH has the important advantage that it allows efficient shortcuts across logical IP subnets, bypassing the routers on the boundary of the logical IP subnets [Smir97]. In order to do that, EARTH proposes a special use of a Multicast Logical IP Subnet (MLIS). A MLIS is a physical ATM network spreading over multiple logical IP subnets that can be served by a single EARTH server. With the support of the EARTH protocol, shortcuts inside the MLIS help achieve the following advantages over MARS not supporting shortcuts: a better performance in terms of join/leave latency and a better efficiency in the multicast distribution tree (MCT) establishment¹ [CoSS98].

The heart of the EARTH protocol is the EARTH server, whose address is well-known to every host in the MLIS. When a host wants to join a multicast group, it sends a registration message called EARTH_join in EARTH terminology that contains the host's IP and ATM address to the EARTH server. The EARTH server uses an address resolution table that is called the Multicast Address Resolution Protocol Table (MARPTable) consisting of multiple member lists to maintain the membership information of the multicast groups. When the EARTH server receives an EARTH_join message from a new receiver, it inserts the receiver's IP and ATM address into the member list of the multicast group the receiver wants to join.

A sender to a multicast group periodically polls the EARTH server for the membership information of the multicast group by sending a query message called EARTH_request in EARTH terminology to the EARTH server in order to get the membership information to update its point to multipoint connection. A message containing the member list of the multicast group is sent back to the sender by the EARTH server as a reply to its EARTH_request. The reply message of the EARTH server is called EARTH_reply in EARTH terminology.

Unlike MARS, EARTH takes a „soft state“ approach. In EARTH, a receiver must periodically send its registration message to the EARTH server to notify that it still wants to receive the multicast traffic. If the EARTH server has not received any registration message from a host for a few refresh periods, the EARTH server assumes that the receiver has left the multicast group and removes its entry from the MARPTable. Next time when the sender queries the

¹ The MCT is the routing tree established by the Inter-Domain Multicast Routing (IDMR) protocols in order to deliver IP datagrams to the receivers.

EARTH server for the receivers' address, it detects that that receiver's address is not contained in the EARTH_reply message and removes it from the point to multipoint connection.

Furthermore, EARTH also provides a mechanism to enable quality of service support, in contrast to MARS not having quality of service support. In EARTH, the MARPTable inside the EARTH server keeps the membership information of a group in a member list that contains the IP and ATM address of the group members. The member list are further divided into several subgroups reflecting the QoS levels the members want to receive.

MIS: The Multicast Integration Server (MIS) architecture is developed by a team of researchers at GMD Fokus and CEFRIEL/Politecnico di Milano. The MIS architecture integrates layer 2 (e.g. ATM) multipoint and Quality of Service with layer 3 (e.g. IP) multicast services and thus enables QoS support for IP multicast over ATM. In the MIS architecture, the EARTH protocol is combined with the RSVP. This architecture achieves the following advantages: increased efficiency, reduced layer 3 processing, reduction of the amount of identical data passing the ATM switches several times, reduced VC consumption and reduced reservation establishment delay in comparison to an architecture with separate address resolution and resource reservation protocol [CoSS98].

In the MIS architecture, the ATM endpoints use a „best effort“ point to point connection as a control channel to exchange EARTH and RSVP messages with the MIS while data is sent via point to multipoint connections with different preferred qualities of service. The MIS is the only point in the ATM cloud where the QoS information is exchanged between the EARTH environment and the RSVP environment and thus requires some modification to RSVP [CoSS97]. RSVP is modified in the MIS architecture in the following way: Path messages are sent from sender(s) to the MIS where they are processed and forwarded to receivers; receivers send their Resv messages to the MIS where they are merged and forwarded to the sender(s). The MIS has a role of a dispatcher of RSVP messages between the sender(s) and receivers [CaCS98].

Figure 4 shows the flow of RSVP messages in the MIS architecture. The RSVP messages are of particular importance in our charging and accounting protocol because they are used, as we will see later, as a means of transport for the charging and accounting information. The EARTH messages are not shown in this figure because they are of little importance in the discussion about the potential design solutions for our charging and accounting protocol presented in Chapter 5.

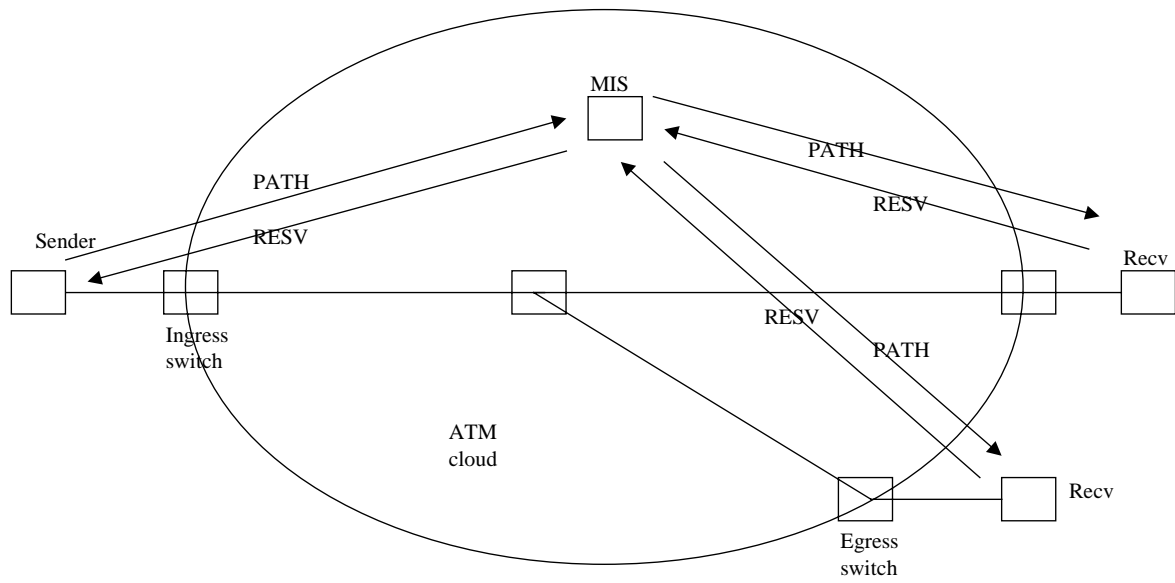


Figure 4: Message flow of RSVP in MIS architecture

3. General Issues of Charging and Accounting in Computer Networks

3.1 A Simple and General Formula

In general, the cost of a connection charged to the users can be proportional to the duration of the connection and the data volume sent through it. In the usage-based pricing model, the charge can also be dependent upon the quality of service of the connection. In [Song97], David Songhurst has proposed a simple and general formula for computing the cost of a connection

$$\text{Cost} = a T + b V + c$$

where T and V are the duration of the connection and data volume traversing it, and a , b , and c are tariff parameters applying to the connection. The tariff parameters are agreed between users and network provider at the start of the data path's establishment. In [Song97], David Songhurst has proposed that the traffic parameters should be static (i.e., they are changed infrequently, typically at intervals in months). The parameter c is the connection set-up cost. It reflects the costs in switching and signaling resources when the data path is established.

However, in some connectionless network technologies, e.g. in classical IP networks, the establishment of the data path is not necessary. In this case, the problem can be solved by agreeing on the traffic parameters at the time when the sender starts sending its data or by using some signaling protocol like RSVP.

In our viewpoint, only tariff parameters applying to unicast data paths should be static. In the case that there are multiple receivers that share the cost of a multicast tree among each other, the charge to a particular receiver depends upon many factors, such as its location in the multicast tree and the total number of receivers. Given the same multicast tree, the more receivers are in a multicast session, the less is the charge to each receiver. Moreover, we believe that a fair pricing scheme should take the location of the receivers into account when the charge to each receiver is computed, e.g. a receiver located three hops away from the sender should pay more than a receiver located two hops away from the sender.

The **Equal Link Split** among **Downstream** members (ELSD) scheme [Herz96a] takes the distance¹ between the sender and a particular receiver into account when computing the charge to this receiver. In the ELSD scheme, the cost of a particular link is shared equally by all downstream receivers of that link; assuming that all these receivers have the same QoS on that link (in section 3.3, we will present a solution proposed by Shai Herzog [Herz96a] to solve the problem when receivers do not have the same QoS on a link). Other receivers are not considered responsible for the cost of that link.

The charge to an individual user may vary with the time because of the dynamic nature of multicast. However, the parameters used to compute the cost of the multicast tree and the pricing scheme applied to share the cost among the users should be static. In his Ph.D. thesis

¹ By distance, we mean both the geographical distance and the hop count a particular receiver is away from the sender.

[Herz96a], Shai Herzog proposed that time can be divided into time intervals, within which the multicast tree can be considered to be static. Thus, the charge to an individual user participating in multicast sessions is static within such time intervals.

The duration of the time intervals must be chosen very carefully. A too large time interval cannot completely reflect the dynamic nature of multicast and hence cannot provide the charging and accounting protocol with sufficiently precise information for computing the charges to the users correctly. Thus, a too large interval would fail to give the users incentives to prefer multicasting to unicasting which is targeted to alleviate the current problems of congestion on the Internet. However, a too small time interval causes unnecessary congestion and exaggerates the current problems of congestion on the Internet. In the prototype implementation of our charging and accounting protocol presented in chapter 5, the time interval is equal to the default refresh period of RSVP (30 seconds).

In our future work, however, the time interval may be changed or implemented in such a way that it can regularly adapt itself to find the optimal trade-off depending on the process of the multicast session and the loaded conditions of the networks, e.g. if the network is congested and the multicast group is rather stable, the time interval should be increased to avoid unnecessary message exchange. Further research and experiments are required in order to find a meaningful mechanism and an optimal trade-off.

In the usage-based pricing model, the tariff parameters must reflect the quality of service, reservation¹ and consumption of network resources along the data path (e.g., a telephone call from Berlin to New York is likely to consume more network resources than a call from Berlin to Paris and hence should be charged higher). Thus, the tariff parameters should be proportional to the distance of the data path. It is also obvious that the tariff parameters must be proportional to the reserved and consumed bandwidth because the charge must reflect the network resource reservation and consumption, and thus gives the users incentives for efficient usage and reasonable behavior.

3.2 Charging and Accounting in ATM Networks

As we will see in chapter 4, it is very difficult to determine the actual routing tree of the point to point or point to multipoint connection in ATM networks. In this section, we propose our solution for determining the tariff parameters of a connection (point to point or point to multipoint) by using the ATM endpoint addresses to approximate the network resource consumption of the connection.

Our solution exploits the hierarchical structure of the ATM addresses and thus avoids difficulties dealing with ATM's routing protocols, such as IISP[ATM94] or PNNI [ATM96]. Although the approximation does not reflect the actual network resource reservation and network resource consumption of the connection, we claim that it should not be considered as a disadvantage but an advantage.

¹ If a user uses a signaling protocol to reserve network resources, he must pay for the reservation even when he does not consume the reserved network resources because the reserved network resources cannot be used by other users.

For example, if a user in Berlin is talking to a friend of him in New York, they do not care much about the actual route taken by the data flow and they always expect to be charged with the same tariff parameters for the point to point connection between Berlin and New York.

3.2.1 Charging and Accounting for Point to Point Connections

If the ATM address of a receiver differs by only a few octets from the ATM address of the sender, it is likely to be connected to the same or a near neighboring switch of the sender's switch and thus it is in the same routing domain of the lowest hierarchy level with the sender. Hence, its point to point connection to the sender is likely to consume or reserve less network resources than a point to point connection between the sender and a receiver in a different routing domain.

Our approach follows a simple approximation: The higher the routing hierarchy level is required to be involved in the data path from the sender to the receiver (i.e., the more digits the receiver's ATM address differ from the sender's ATM address), the more network resources the data path is likely to consume and thus the more the users (i.e., sender or receiver) should pay for the connection, i.e. the greater the tariff parameters are. We propose that network providers should have a table containing the tariff parameters as functions of the hierarchical routing level required to be involved to resolve the routing for the connection.

Examples:

- If the ATM addresses of the sender and receiver differ by fewer than seven octets, then the sender and receiver are in the same routing domain of the lowest hierarchy level. In this case, the users (i.e., sender or receiver) is charged with a low cost because the network resource consumption of the connection is approximately small.
- If the ATM addresses of the sender and receiver differ by eight octets, then the sender and receiver are in the same group of the second lowest hierarchy level. In this case, the users (i.e., sender or receiver) is charged with a higher cost because the connection is likely to consume more network resources compared to the connection within the same routing domain of the lowest hierarchy level.

3.2.2 Charging and Accounting for Point to Multipoint Connections

If the ATM addresses of two receivers differ by a few octets, then the receivers are likely to be in the same routing domain and have some links in common in the point to multipoint connection and the branching point in the point to multipoint connection is likely to be somewhere in the same routing domain with these receivers. We suggest that these receivers share the cost of the path from the sender to the branching point. Each receiver is additionally charged with the cost of the path from the branching point to itself. Furthermore, we can recursively consider the branching point as an „endpoint“ in order to share the cost of the path from the sender to the branching point in the case that there are other branching points upstream. If two users connected to the same switch share the cost of the point to multipoint connection, the tariff parameters they have are likely to be half of the tariff parameters applying to a point to point connection from a single user to the sender.

Examples:

- Let's assume that we have a sender and two receivers, and their ATM addresses are
 sender: 0x39.276f.31.0001ef.0000.0401.7002.002048063b8c.00
 receiver 1: 0x39.276f.31.0001ef.0000.0401.7004.002048163b8c.00
 receiver 2: 0x39.276f.31.0001ef.0000.0401.7004.002048263b8c.00.

In this case, the ATM address of the branching point of the point to multipoint connection must be something like 0x39.276f.31.0001ef.0000.0401.7004.xxxxxxxxxxxx.xx, say 0x39.276f.31.0001ef.0000.0401.7004.002048063b8c.00. Thus, each receiver is charged with the half of the cost of the point to point connection from the sender to the branching point and the cost from the branching point to itself.

- Let's assume that we have a sender and three receivers, and their ATM addresses are
 sender: 0x39.276f.31.0001ef.0000.0400.7002.002048063b8c.00
 receiver 1: 0x39.276f.31.0001ef.0000.0401.7003.002048063b8c.00
 receiver 2: 0x39.276f.31.0001ef.0000.0401.7004.002048163b8c.00
 receiver 3: 0x39.276f.31.0001ef.0000.0401.7004.002048263b8c.00.

In this case, we have two branching points. The ATM address of the downstream branching point must be something like 0x39.276f.31.0001ef.0000.0401.7004.xxxxxxxxxxxx.xx, say 0x39.276f.31.0001ef.0000.0401.7004.002048063b8c.00, and the ATM address of the upstream branching point must be something like 0x39.276f.31.0001ef.0000.0401.xxxx.xxxxxxxxxxxx.xx, say 0x39.276f.31.0001ef.0000.0401.7002.002048063b8c.00. The cost of the path from the sender to the upstream branching point must be equally shared by all the three receivers and the cost of the path between the branching points must be shared equally by the first two receivers. Each of the first two receivers are additionally charged with the cost of the path from the downstream branching point to itself. The third receiver has to pay the cost of the path from the upstream branching point to itself.

Figure 5 illustrates the idea of our proposal and the examples presented above.

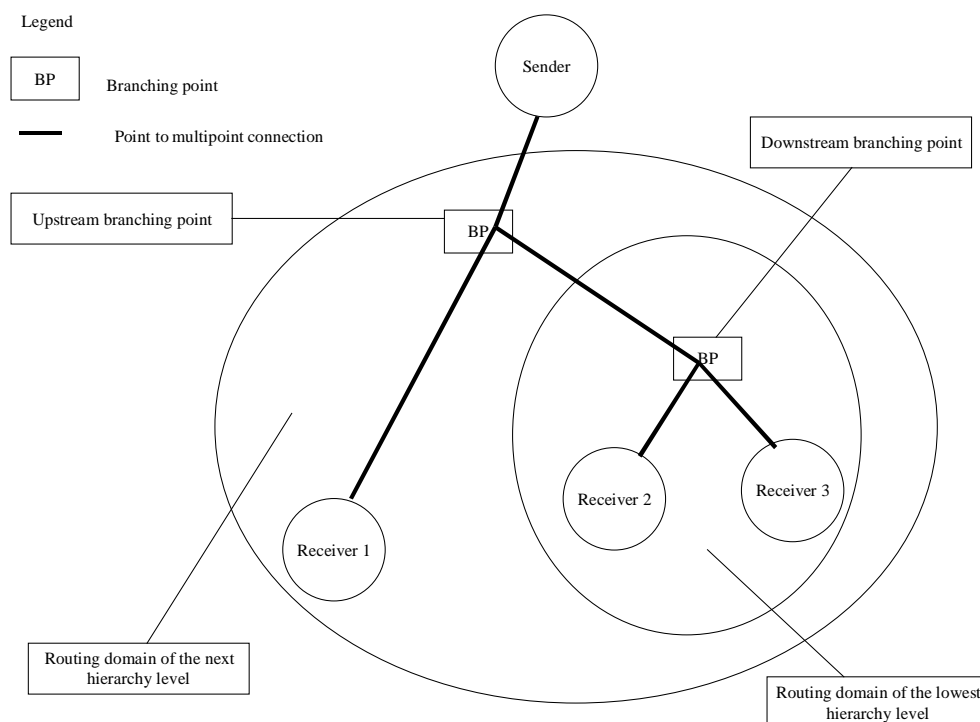


Figure 5: Sharing the cost of a point to multipoint connection in ATM networks among the receivers.

3.3 Charging and Accounting in Conjunction with RSVP and IP Multicast

In section 3.2, we have proposed a solution of how to compute the cost of a connection (point to point or point to multipoint) and how to compute and share the cost of a point to multipoint connection among the receivers in ATM networks. This solution cannot be applied to the Internet because the Internet does not have a hierarchical structure of addresses and routing like the ATM technology¹. In chapter 5, we will review a solution of Shai Herzog for how to share the cost of the multicast tree among the users on the Internet.

It is even more difficult to share the cost of the multicast tree among the receivers on the Internet when qualities of service support is provided (i.e., users can choose the QoS level they desire to have².) The problem we observed in section 3.2 is how to share the cost of the multicast tree among the receivers with a homogenous QoS level. In this section, we will take a look at another problem, namely how to determine the shared charge of the cost of a link for the users who are connected to the same link but desire to have different qualities of service. In the context of RSVP, if a router receives several Resv messages reflecting different desired QoS levels from its receivers or downstream routers and all of the reservations have the same reservation style³, it will try to merge the reservations and sends a Resv message with the highest QoS level upstream. In Shai Herzog's solution presented here, we simply make the assumption, which appears rather natural and reasonable, that the cost of the link is the cost computed for the highest QoS level.

In his Ph.D. thesis [Herz96a], Shai Herzog has proposed a solution for the problem above as follows:

- If the set of qualities of service is fully ordered, every user has to share equally the cost computed for the lowest QoS level. Users who desire to have a higher QoS level have to share equally the incremental cost computed for all levels less than or equal to their QoS level.
- If the set of qualities of service is not completely ordered, an ad-hoc metric, such as the cost computed for the QoS levels, can be used to fully order the QoS levels and the cost of the link can be shared as described above in the case that the set of qualities of service is fully ordered.

Examples:

- Let's assume that we have two receivers connected to the same link. The first receiver has a reservation of 3 Mbit/s and the second receiver has a reservation of 5 Mbit/s. The costs for the reservation of 3 Mbit/s and 5 Mbit/s are \$3/day and \$4/day⁴, respectively. According to the context of RSVP and the assumption we made above, the highest reservation of the two

¹ This is not true for IPv6. However, we expect that it will take years until IPv4 is fully supplanted by IPv6.

² We do not have this problem in ATM networks because all the leaves of a point to multipoint connection have the same quality of service.

³ The reservation style specifies the selection of the sender whose data traffic may receive the reserved QoS.

⁴ The prices here are taken only as examples and do not necessarily reflect the prices in "real life".

receivers on the link is 5 Mbit/s and costs \$4/day. In this case, the two receivers have to share equally the costs for the lowest reservation, namely 3 Mbit/s, and the second receiver has to pay the incremental cost for the higher reservation. The first receiver has to pay \$1.5/day and the second receiver has to pay \$2.5/day.

- Let's assume that we have three receivers connected to the same link. The first receiver has a reservation of 3 Mbit/s, the second receiver has a reservation of 5 Mbit/s, and the third receiver has a reservation of 6 Mbit/s. The costs for the reservation of 3 Mbit/s, 5 Mbit/s and 6 Mbit/s are \$3/day, \$4/day and \$5/day, respectively. According to the context of RSVP and the assumption we made above, the highest reservation of the three receivers on the link is 6 Mbit/s and costs \$5/day. In this case, the three receivers have to share equally the costs for the lowest reservation, namely 3 Mbit/s. The second and the third receiver have to share equally the incremental cost for the reservation of 4 Mbit/s, namely \$4/day - \$3/day = \$1/day. Finally, the third receiver has to pay the incremental cost of the reservation of 5 Mbit/s. The first receiver has to pay \$1/day, the second receiver has to pay \$1.5/day and the third receiver has to pay \$2.5/day.

Figure 6 illustrates the idea and the examples discussed in this section.

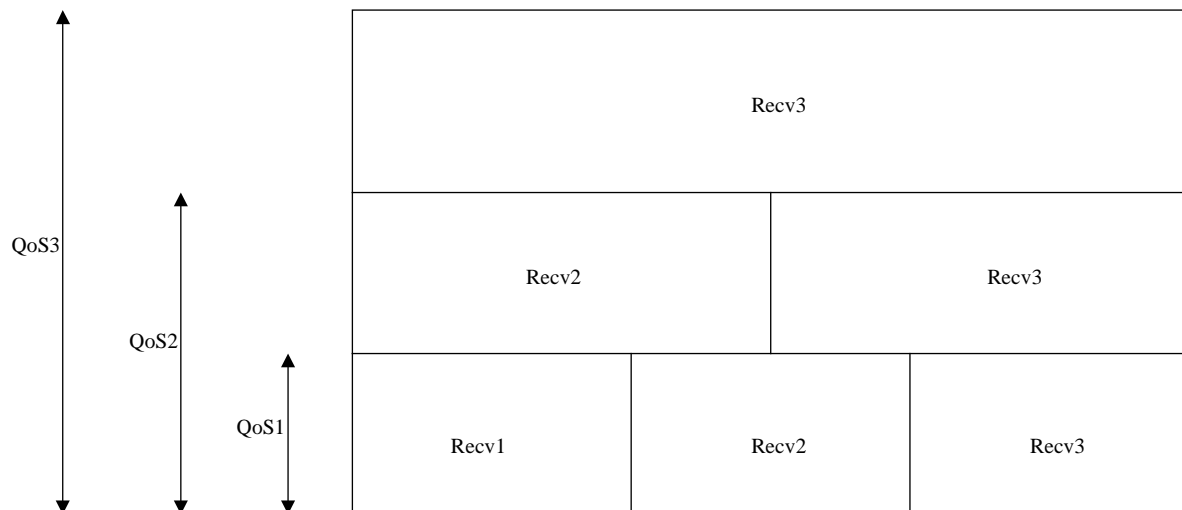


Figure 6: Sharing the cost of a link of receivers having different QoS levels.

4. Protocol Requirements

In this chapter, we will list all the requirements we consider as meaningful and thus should be satisfied by our charging and accounting protocol (CAP). The following requirements for a charging and accounting protocol are defined:

Fairness: Fairness is very important in our point of view. Fair cost sharing belongs to our most important design goals. We contend that the **Equal Link Split** among **Downstream** members (ELSD) scheme [Herz96a] satisfies our requirement of fairness and is a good candidate to be implemented in our charging and accounting protocol. In the ELSD scheme, the cost of a particular link is shared equally by all downstream receivers of that link. All other receivers are not considered responsible for the cost of that link.

Edge Pricing: A new pricing paradigm named „edge pricing“ is proposed in [Shen96] by Scott Shenker et al. The edge pricing paradigm states that a charging and accounting protocol should support local pricing policy (i.e. it should not support any particular pricing policies and must be open to the choice of pricing policy) because every network provider may choose to have his own pricing policy. We require our charging and accounting protocol to be fully in line with the edge pricing paradigm.

However, we do not completely agree with the first basic approximation advocated by edge pricing that the charge to the users is computed based only on the expected congestion conditions along the path and not on actual measurements. In our charging and accounting protocol, the cost of the data path (unicast or multicast) is computed based on the actual reservation and consumption of network resources.

Reliable Communication of the Charging Information: In this work, we try to tackle the issue of charging and accounting for the „costs“ of network resources that we call „technical costs“. In general, the technical cost is not the cost which is finally charged in money because the charged cost is also based on marketing, regulatory and many other issues. However, it is very likely that the „technical costs“ will be taken into account when the charged cost is computed. Thus, a charging and accounting protocol should be reliable to serve as a useful input to computing the charged cost. Our protocol takes a „soft state“ approach and timeout mechanism and fulfills the requirement of reliability.

Furthermore, the CAP modules maintain log files of charging and accounting information in order to allow for the case of crashing and losing information. In the case that a CAP module crashes, the network operator can use the log files to compute the charges by hand.

Besides, we also provide our charging and accounting protocol with a simple algorithm for acknowledging and re-transmitting the charging information in order to enable a reliable exchange of charging information between the CAP modules.

Scalability: Scalability is one of the most important requirements that our protocol has to satisfy. A charging and accounting protocol with a poor scaling property is obviously of little use. This feature is especially important because many researchers and engineers believe that a large number of new applications in the near future will rely on multicast transmissions and/or a reservation protocol like RSVP. Hence, we want our charging and accounting protocol to be

at least as scalable as IP multicast and/or RSVP so that it can be used for charging in these new applications.

Protocol Independence: Although it was the original target of our work to design a protocol that provides the MIS architecture with charging and accounting ability, e.g. a protocol that runs in a specific architecture. It is our ambition to design a general charging and accounting protocol that can run in conjunction with any signaling protocol (e.g., RSVP) on any network technologies. This is only possible by a well-defined interface between the signaling and the charging and accounting protocol.

Charging and Accounting inside the Network: In our viewpoint, a network provider should never faithfully trust the senders and receivers, i.e. users, participating in unicast or multicast sessions. Therefore, the task of collecting the charging and accounting information and computing the charges to the users must be done somewhere inside the network and not in end systems (i.e., never in senders or receivers).

Simplicity: It is widely agreed that networks must be simple in order to be fast. This is especially true for routers. Thus, a complicated charging and accounting protocol that conveys charging and accounting information throughout the network would degrade the performance of the network. In fact, we believe that every sane charging and accounting protocol has a simple and intuitive nature.

Security: Although security is a very important issue for a charging and accounting protocol, we do not address this topic in this work. In our prototype implementation, the charging and accounting information is exchanged unencrypted in RSVP messages. Hence, in theory, a user could pretend to be a downstream router or another user and intentionally give his upstream router wrong information. However, we believe that the charging and accounting information can be transmitted using the same method applied for the RSVP authentication information. Refer to [Bake96] for more details. In future work, we will extend our charging and accounting protocol with some authentication mechanism to prevent users from spoofing their providers.

5. Architecture of the Charging and Accounting Protocol

5.1 Terminology

In this section, we introduce the terminology which we will use throughout the next chapters in order to avoid misunderstanding.

- The directional terms „upstream“ vs. „downstream“, „ingress“ vs. „egress“ are used with respect to the direction of the data flow.
- Ingress node is a router or, generally speaking, an interworking unit where the data traffic enters a network.
- Egress node is a router or, generally speaking, an interworking unit where the data traffic exits a network.

5.2 Service Scenarios and Potential Solutions

Sender-paying and receiver-paying are two models we try to address in this work because we believe that any other pricing model can be considered as a combination of these two models. In this section, we will present some service scenarios and try to find all potential solutions for them. We will consider the advantages and disadvantages of these solutions and will choose the solution we believe to be the best of them.

The service scenarios in this section are presented in the context of the MIS architecture because this work is originally intended to enhance the MIS architecture by the capability of charging and accounting. All of our solutions have a CAP server that is a workstation responsible for debiting the hosts participating in multicast sessions. The solutions presented in this section are based on the MIS architecture where IP meets ATM so that they are “mixed” because we want to exploit the feature of the MIS architecture. In section 5.3 we will present a general charging and accounting protocol that runs in “pure” IP networks.

The CAP server can also be in charge of computing the charges to those hosts. However, this task can be shifted, depending upon the service scenario and the architecture of the solutions, to another point inside the network. In this case, the charging information has to be transferred from that point to the CAP server where the hosts are debited.

An obvious way to transfer the charging information is to use the reliable transport service of TCP. However, when CAP runs in the MIS architecture (i.e., the MIS architecture is extended by the CAP in order to have a charging and accounting capability), the MIS turns out to be a suited location for the CAP server. This is because the CAP server can use RSVP as a transport protocol to communicate the charging and accounting information with the CAP modules located in routers or hosts.

In the MIS architecture, all ATM endpoints use a point to point connection to exchange EARTH and RSVP messages with the MIS. Thus, the aggregation of the MIS and the CAP server reduces the complexity and VC consumption. Moreover, the MARPTable can be extended to contain charging and accounting information besides the membership information

of the users¹. The MIS with capability of charging and accounting is called Multicast Integration Charging and Accounting Server (MICAS).

5.2.1 Service Scenario 1: one ATM Network in Combination with MICAS

In the MIS architecture, the ingress node (i.e., the router connected to the sender host) and the MICAS are two candidates for the task of computing the cost of the multicast tree. This is because they have the ATM address of the receivers and can use this information to compute the cost of the multicast tree (in this service scenario, the cost of the multicast tree is the cost of the point to multipoint connection). Receivers cannot easily compute the cost of the multicast tree because they do not have the whole information about the topology of the multicast tree.

We argue that the switches inside the network are not suitable for this computing task because they must be kept as simple as possible in order to be fast. Computing the cost of the multicast tree would unnecessarily consume CPU resources of the switches and also make them more complicated, thus resulting in a performance degradation of the switches. Hence, it would be a bad design to deploy a charging and accounting protocol that runs between the switches on the ATM layer. In this section, we simply assume that the ingress node or the MICAS can somehow use the whole knowledge of the multicast tree topology to compute its cost and the shared charges of the receivers if the receiver-paying model is applied. Refer to chapter 3 for more details about how to compute and share the cost of connection in ATM networks² (point to point or point to multipoint).

5.2.1.1 Sender-paying Model

5.2.1.1.1 MICAS is in Charge of Computing the Cost of the Multicast Tree

In this solution, the MICAS uses the information of the MARPTable compute the cost of the multicast tree and charges this cost to the sender. The MICAS can use RSVP as a transport protocol to transfer the charging information to the sender. It encapsulates the charging information in a Resv message and sends it to the sender. By this way, the sender can be informed about the cost of the multicast tree it has incurred.

5.2.1.1.2 Ingress Node is in Charge of Computing the Cost of the Multicast Tree

After having computed the cost of the multicast tree, the ingress node can use the service of a transport protocol to transfer the charging information to the MICAS. The MICAS then uses this charging information to charge the sender. Although RSVP is a layer 3 protocol, it could be a good candidate for the task of transporting the charging information from the ingress node to the MICAS because the ingress node would only have to embed the charging information in RSVP messages and would have RSVP deliver it to the MICAS. This solution

¹ Another alternative is to use a separate table to contain the charging and accounting information. This alternative is more complex than the first one and has two tables of similar structure. However, it clearly separates the charging and accounting protocol from the MIS architecture and thus increases the independence of the CAP.

² Note that our proposals in Chapter 3 are not the only way to compute the cost of a connection and share it among the users (in case of multicast tree of point to multipoint connection with receiver-paying model). The method to compute the cost of a connection and share them among the users may be modified in our future implementation.

does not have any advantages over the previous one. Moreover, some mechanism must be implemented to make the transfer of charging information reliable because RSVP messages are exchanged at IP layer (layer 3) without any error-control mechanism. Thus this solution is more complicated than the previous one because computing of the charge is done in the ingress node, then the charging information has to be transferred to the MICAS where the charging and debiting finally takes place in the MICAS. The solution is even worse if the ingress node does not support RSVP. In this case, the ingress node has to use the service of another transport protocol to transfer the charging information to the MICAS. There are four potential solutions for this problem:

- The ingress node could use the ATM layer to transfer the charging information to the MICAS. However, the ATM layer is not suitable for this task because it does not offer a reliable transport service and accepts only 48-byte datagrams. Thus, the ingress node would have to implement an adaptation layer over ATM to have a reliable transport service that is not limited to 48-byte datagrams. We contend that this solution is bad because it is complicated and inefficient.
- The ingress node could use IP as a transport protocol to transfer the charging information to the MICAS. However, some mechanism must be implemented to ensure a reliable communication of the charging information because IP is unreliable. This solution is similar to the solution using RSVP as a transport protocol if RSVP is implemented in the ingress node.
- The ingress node could also use TCP a transport protocol because it provides a reliable transport service. However, we contend that this is a messy and complicated solution because the ingress node uses the signaling information (i.e., the information about the topology of the multicast tree) of the ATM layer (layer 2) to compute the cost of the multicast tree and then uses the service of a transport layer (layer 4) to transfer the charging information to the MICAS where the sender is finally charged. The ingress node can also use the reliable service of TCP to transfer the charging information to the sender to inform the sender about the cost it has incurred.
- Another potential solution for this problem is to use SNMP¹ to get the routing information of the ingress node from a dedicated computer (e.g., the MICAS). This computer then uses the routing information of the ingress node to compute the charge to the sender. After having computed the charge to the sender, it has to use the reliable transport service of TCP to transfer the charging information to the MICAS. This solution is also complicated and inefficient because the information must flow from the ingress node to a dedicated computer and then from there to the MICAS. This solution assumes that the ingress node supports SNMP.

All in all, we consider that the ingress node is not suitable for the task of computing the cost of the multicast tree because all of its potential solutions are complicated and inefficient.

5.2.1.2 Receiver-paying Model

In this section, we simply assume that the MICAS or the ingress node can somehow apply the ELSD scheme to compute a fair shared charge to the receivers participating in a multicast

¹ SNMP: Simple Network Management Protocol

session. Refer to chapter 3 for more details about how to compute the shared charges for multiple receivers.

In theory, the ingress node or the MICAS are required to compute the new shared costs for the receivers every time the topology of the multicast tree changes (e.g., some new receivers join or some receivers leave the multicast session). However, this is impossible in practice because of the dynamic and distributed nature of multicast. We propose that time be divided in small intervals, within that the multicast tree are considered static. Based on this proposal, the ingress node or the MICAS periodically updates its knowledge about the multicast tree, and uses this knowledge to compute the fair shared charges to the receivers. Refer to chapter 3 for more details.

5.2.1.2.1 MICAS is in Charge of Computing the Cost of the Multicast Tree

In this solution, the MICAS uses the membership information in the MARPTable to compute the fair shared charges to all receivers and then debits the receivers with these charges. The MICAS informs the receivers about their charges by encapsulating the charging information in Path messages and sends these messages downstream to the receivers (in the MIS architecture, all receivers inside the ATM cloud are the next downstream hops of the MIS).

5.2.1.2.2 Ingress Node at the Sender is in Charge of Computing the Cost of the Multicast Tree

In this solution, the ingress node computes the fair shared charges to all receivers and uses a reliable service of a transport protocol to transfer the charging information to the MICAS. The MICAS then uses this information to charge the receivers. The MICAS also forwards the charging information encapsulated in Path messages to the receivers to inform them about the costs they have incurred. This solution is rather complicated (see also 5.2.1.1.2).

5.2.2 Service Scenario 2: Multiple Legacy LANs Attached to an ATM network

In this service scenario, we can divide the cost of the multicast tree into two parts, the cost on the ATM network and the cost on the legacy LANs. The cost on the ATM network can be determined and charged to the sender or receivers as described in 5.2.1. In this section, we simply assume that the costs on the legacy LANs can somehow be determined by some special devices which we call CAP clients.

These CAP clients are placed on the legacy LANs and take some measurements to determine the receivers' consumption of network resources¹. They communicate with their CAP server to report the charging information to it. The CAP server will use this information to charge the sender or receivers. In large networks, there may be many CAP servers. The CAP clients communicate to the CAP server which is next to them.

Another solution is to let the CAP clients send the charging and accounting information to the CAP server inside the ATM network (the MICAS). However, this solution can lead to a message implosion in a large network (consider the case of an ATM network serving as a backbone and many legacy LANs attached to it). We propose that some merging techniques

¹ We do not delve into finding a solution for this problem and simply consider that this problem goes beyond the scope of this work.

must be used to merge the charging and accounting information at branching points of the multicast tree in order to avoid unnecessary network congestion¹ if this solution is chosen.

An obvious way for the CAP clients to report the charging and accounting information to the MICAS is to encapsulate this information as an object in Resv messages and send these messages upstream to the MICAS. Having the costs incurred by the sender on the ATM network and on the legacy LANs, the MICAS can compute the sum of them and charge it to the sender or receivers.

5.2.2.1 Sender-paying Model

In the sender-paying model, there are no necessary changes for the charging of the costs on the ATM network. We have two alternative solutions for charging the costs on the legacy LANs to the sender:

- CAP clients send the charging information to the MICAS. The MICAS charges the costs on the legacy LANs to the sender.
- There are other CAP servers on the legacy LANs responsible for the task of charging. CAP clients send the charging information to these CAP servers. The CAP servers must run another protocol to communicate the charging information with the MICAS in order to produce only one bill for the sender host².

5.2.2.2 Receiver-paying Model

According to the ELSD scheme, the cost of a particular link on the ATM network must be shared equally by all of its downstream receivers. The routers connecting the legacy LANs to the ATM network must know how many downstream receivers they have and send this information upstream to the MICAS or to the ingress node at the sender. The routers can do this by encapsulating the information in Resv messages and send them upstream. The MICAS or the ingress node at the sender use this information to build the topology of the multicast tree and compute the fair shared charge to the receivers. Again, we argue that the ingress node is not suitable for the task of computing the charge to the receivers (see 5.2.1.1.2).

If the ingress node is in charge of computing the shared charges on the ATM network, it must transfer the charging information to the MICAS. The MICAS uses the charging information to debit the receivers with their shared charge. It also encapsulates the charging information in Path messages and sends them downstream to the receivers to inform them about the costs they have incurred on the ATM network. This solution is simple because the MICAS is the only charging server in the architecture and it does not require any communication with another charging server. However, it is not optimal because the charge to each receiver may not be the same. The charging information for each receiver must be sent to it explicitly and not to the other receivers. This is not fully in line with the RSVP and can cause a message implosion in the case of a large multicast session with hundreds of receivers on legacy LANs attached to an ATM network.

¹ If there is a choice, one should choose to create more or larger messages with a small time-to-live (TTL) value rather than fewer or smaller messages with a large TTL value (private communication with Dr. Georg Carle).

² This solution probably requires a synchronization between the MICAS and the CAP servers and goes beyond the scope of this work. A server synchronization protocol has been proposed in [Luci97].

Another solution is to let the MICAS compute the sum of charges to all downstream receivers of the routers connecting the legacy LANs to the ATM network. The MICAS informs the routers about the total charge of their downstream receivers by encapsulating the charging information in Path messages and send them downstream to the routers (in MIS architecture, the routers connecting the legacy LANs to the ATM network are the next downstream hops of the MICAS). When a router receives the charging information from the MICAS, it can also apply the ELSD scheme to compute the fair shared charge for each of its downstream receivers and charges it to them.

5.3 General Charging and Accounting Protocol

5.3.1 Design issues

In the previous section, we discussed different potential solutions for charging and accounting in the MIS architecture, their advantages, and disadvantages. During the research of this work, we also searched for a general solution that can be deployed on other network technologies besides the MIS architecture.

In this section, we present a general charging and accounting protocol that we believe to be deployable in conjunction with RSVP on any network technologies. The protocol presented in this section is based on the work of Shai Herzog [Herz96a]. However, Shai Herzog only concentrated on a solution for the receiver-paying model. In this section, we extend the work of Shai Herzog by presenting a solution for the sender-paying model and a mechanism enabling a reliable communication of charging and accounting information between the CAP modules. We will also provide a discussion about the design alternatives.

Note that in the MIS architecture, the MIS is the direct downstream neighbor of the sender and upstream neighbor of the receivers or multicast routers placed on the edge of the ATM network. Thus, the charging and accounting protocol presented here is an extension, or better a generalization, of the solution we presented and favored in the last session.

Our CAP is a protocol supporting charging and accounting that runs in conjunction with RSVP or a similar signaling protocol. Each RSVP node (i.e. sender, router or receiver) is extended by a small module that provides the support for charging and accounting. The communication between the RSVP module and the CAP module takes place in a form of API calls and will be discussed in more details later. Charging and accounting information is encapsulated as objects in RSVP messages that are exchanged between an RSVP node and its next upstream or downstream hops. The encapsulation of the charging and accounting information in RSVP messages uses the TLV¹ concept; new CAP objects can be easily defined in this way if needed.

Our CAP takes the same „soft state“ approach and time-out mechanism like RSVP. When a CAP module has not received any messages from its downstream hop for a period of time, it assumes that all the downstream receivers of that hop have already left the multicast group and does not charge them anymore (if the receiver-paying model is used). It updates its total number of receivers and computes the shared charges to its remaining downstream receivers.

¹ TLV: Type, length, value.

If the sender-paying model is used, the CAP module stops charging the cost of the link connecting to the timed out downstream hop to the sender.

An easy solution for implementing the „soft state“ approach and time-out mechanism is to enable the synchronization between RSVP and CAP via the RSVP-CAP interface introduced below. In this solution, the RSVP has to notify the CAP module when it prunes a timed out downstream hop. Another solution is to implement a separate and independent timeout mechanism for the CAP module. However, this solution needs a careful design to avoid the loss of synchronization between the RSVP and CAP modules. The first solution is chosen to be implemented in our prototype implementation because of its simplicity and efficiency.

A CAP module also has to maintain some log files for the objects sent to its upstream or downstream nodes. This allows for the case that the computer or router hosting the CAP modules crashes. In this case, the network administrator can use the log files to compute or correct the charges by hand afterwards.

The RSVP-CAP interface is based on the principle of the RSVP-LPM¹ interface [Herz96a, Herz96b, Rap98a]. The RSVP-CAP interface is defined as a set of services (API style) that are provided by the CAP module to the RSVP module. These services are divided into two main categories: input and output. Incoming CAP objects encapsulated in RSVP messages trigger RSVP to call the CAP input services while outgoing RSVP messages require RSVP to call the CAP module's output services to create outgoing CAP objects which are embedded in RSVP messages and transferred to the upstream or downstream hops. The RSVP-CAP interface also enables the synchronization between these two protocols. When RSVP establishes a reservation for a downstream hop, it notifies the CAP via the RSVP-CAP interface, triggering the CAP to start its charging and accounting task for the downstream hop. When RSVP releases the network resources reserved by a downstream hop (i.e., RSVP has received a ResvTear message from that downstream hop or the RSVP "soft state" of that downstream hop has timed out), it has to notify the CAP via the RSVP-CAP interface and stops the CAP charging the downstream hop.

Figure 7 illustrates the RSVP-CAP interface and the method the CAP modules use to exchange the charging and accounting information piggybacked in RSVP messages with their neighbor hops. Note that the picture is not intended to illustrate the functionality of RSVP. Thus, the features of reservation and admission are not shown in the picture. A well-defined interface between RSVP and CAP also helps to have only little modification in RSVP code; this is one of our targets.

¹ LPM: Local Policy Module

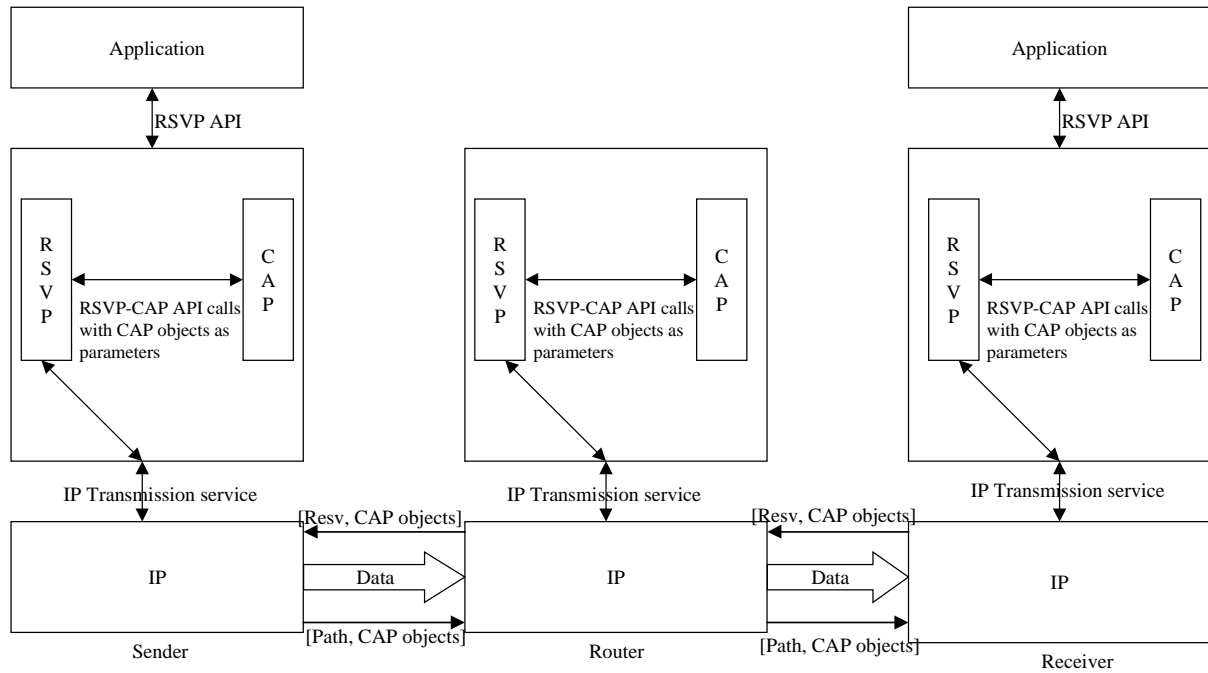


Figure 7: RSVP-CAP interface

CAP requires that each RSVP node know the cost of its downstream links. An RSVP node can be easily extended by a small module which carries out some measurements to determine the cost of the links. Each CAP module must have at least one CAP server where the debiting of the users actually takes place. In a large network, there may be multiple CAP servers (e.g., one CAP server for an IP subnet) which may communicate with each other by means of another protocol. However, this issue goes beyond the scope of our work.

It is a design issue whether the CAP modules should employ merging techniques like RSVP to aggregate the charging and accounting information or they should sample the information, such as how many downstream receivers they have and how much the costs of their downstream links are, and simply send this information to their CAP server.

The first design alternative makes CAP well scaleable, conformant to the edge pricing principle and is also fully in line with the RSVP. In the first design alternative, the CAP modules located on the RSVP routers have to run some algorithms to compute the charges to the receivers. This imposes extra work on the routers and might reduce their performances. However, we argue that this extra work is only a few simply arithmetic operations in our design and thus its influence on the performances of the routers is negligible.

In the second design alternative, the routers do not have to compute the charges to the receivers. They only sample the charging and accounting information and send it to a CAP server. However, all the charging and information must be sent to the same CAP server in this case because this CAP server has to have the entire knowledge of the topology of the multicast tree and its link costs in order to apply the ELSD scheme presented in Chapter 3 and 4 to compute the shared charges to the receivers.

The second design alternative does not impose extra work on the routers and shifts this work to the CAP server. However, this design alternative might not scale well in large multicast sessions. In the case that a host in Berlin is receiving multicast traffic from a sender in New

York, the router in Berlin has to send the charging and accounting information to the CAP server in New York and causes unnecessary traffic on the Internet. Moreover, the receivers might wish to be informed about the costs they are charged. Thus, the CAP server might have to send the information about the charges downstream to the receivers. In the ELSD scheme, the charge to a receiver depends upon the topology of the multicast tree and where the receiver is located. Thus, the charge might not be the same for all receivers. The CAP server would have to send to each receiver a particular CAP object containing the charging information to inform the receiver about the charge to it. This would lead to more congestion on the Internet.

The second design alternative also does not fully conform to the edge pricing principle. If the network providers in Berlin and New York are not the same, i.e. the multicast tree spreads over multiple networks of different network providers, this design alternative would be very cumbersome because each network provider might choose to have its own pricing structure (e.g., the network provider in Berlin might choose to use a flat pricing scheme, while the network provider in New York enforces the usage-based pricing model). Besides, a network provider would not like to let its competitors know about its pricing scheme and the topology of its networks.

5.3.2 Protocol specification

Due to the discussion about design alternatives above, we believe that the first design alternative is better. Following, we will present a protocol specification for this design alternative.

In our charging and accounting protocol, a CAP module exchanges different types of information with its direct upstream and downstream neighbors. We call a piece of information of a particular type a CAP element. A single CAP object encapsulated in an RSVP message contains an RSVP object header and multiple CAP elements. Each CAP element, in turn, has an element header to help the CAP module to determine what type of information is conveyed.

A CAP object has an RSVP object header because it must be recognized and identified by RSVP. The RSVP header for a CAP object is defined as follows:

```
struct cap_header
{
    int16_t m_length;
    u_char m_class;
    u_char m_type;
};
```

m_length specifies the length of the CAP object. *m_class* and *m_type* specify are used as object identification for CAP objects [Brad97]. In our prototype implementation, *m_class* and *m_type* are chosen to be 0x14 and 1¹, respectively.

The header of a CAP element is defined as follows:

¹ There is no specific reason why these values are chosen. We chose these values simply because they are currently not used by other RSVP objects.

```

struct cap_elem
{
    int16_t length;
    u_char cap_elem_type;
    int seqno ;
    /* (char*) data (if needed) depending on the element type is attached here */
};

```

length specifies the length of the CAP element.

cap_elem_type specifies the type of the CAP element.

seqno specifies the sequence number of the CAP element; and will be discussed later.

Specific data (if needed) depending on the type of the CAP element follows the header.

Currently, the following CAP elements are defined:

- Paying model

The “paying model” CAP element contains a CAP element header and an integer specifying the paying model pertaining to an RSVP session.

This CAP element is encapsulated in Path messages and sent downstream. It defines the paying model (sender-paying or receiver-paying) used in the RSVP session. The CAP modules use this element to initiate their internal variables and counters appropriately.

- Downstream receivers

The “downstream receiver” CAP element contains a CAP element header and an integer specifying the total number of downstream receivers of an RSVP node.

This CAP element is encapsulated in Resv messages and sent to the next upstream hop. It is used in the receiver-paying model to specify the total number of downstream receivers of an RSVP node. The sequence number in the CAP element header is used as the acknowledgement for the CAP elements containing the charging information.

- Upstream charge

The “upstream charge” CAP element contains a CAP element header and a floating point specifying the total charge of the multicast subtree upstream of an RSVP node. This charge is to be shared among the downstream receivers of an RSVP node.

This CAP element is encapsulated in Path messages and sent to the next downstream hops. It is used in the receiver-paying model to specify the shared charge of the cost of the part of the multicast tree from the sender down to the RSVP hop sending this object. This shared charge is charged to all the downstream receivers of the downstream RSVP hop and must be shared equally by them. The sequence number contained in the header of the CAP element is used in the context of the “Go Back n” protocol to enable a reliable communication of the charging information between the CAP modules.

- Next hop

The “next hop” CAP element contains a CAP element header and an IP address specifying the receiver of the CAP object. This is necessary in the receiver-paying model when there are multiple receivers or routers on a broadcast medium network (e.g., legacy LAN). Due to their

different QoS or different number of downstream receivers in case of routers, they might receive different charges.

In receiver-paying model, this CAP element must be the first element of a CAP object encapsulated in a Path message. If the first CAP element of a CAP object encapsulated in a Path message is not a “next hop” CAP element or if the IP address contained in the “next hop” CAP element does not match any of its interface’s IP address, the CAP module simply discards the CAP object. This is a design matter to keep the CAP module simple and efficient.

- **Downstream charge**

The “downstream charge” CAP element contains a CAP element header and a floating point specifying the total charge of the multicast subtree downstream of an RSVP node.

This CAP element is encapsulated in Resv messages and sent to the next upstream hop. This element is used in the sender-paying model and specifies the total cost of the part of the multicast tree downstream of an RSVP hop. The sequence number contained in the header of the CAP element is used in the context of the “Go Back n” protocol to enable a reliable communication of the charging information between the CAP modules.

- **Downstream charge acknowledgement**

The “downstream charge acknowledgement” CAP element contains only a CAP element header.

This element is encapsulated in Path messages and sent to the next downstream hop. It is used in the sender-paying model to acknowledge the receipt of a „downstream charge“ element from the downstream hop. The sequence number contained in the CAP element header is used as an acknowledgement for the correctly received “upstream charge” CAP elements.

See section 3 of [Brad97] for more details about RSVP objects and encapsulation of objects inside RSVP messages.

Again, we try to address the two main paying models, sender-paying and receiver-paying model. For each of these models, our CAP uses a separate mechanism.

Receiver-paying Model

The message sequence chart in figure 8 shows the flow of the RSVP messages and CAP objects in the receiver-paying model.

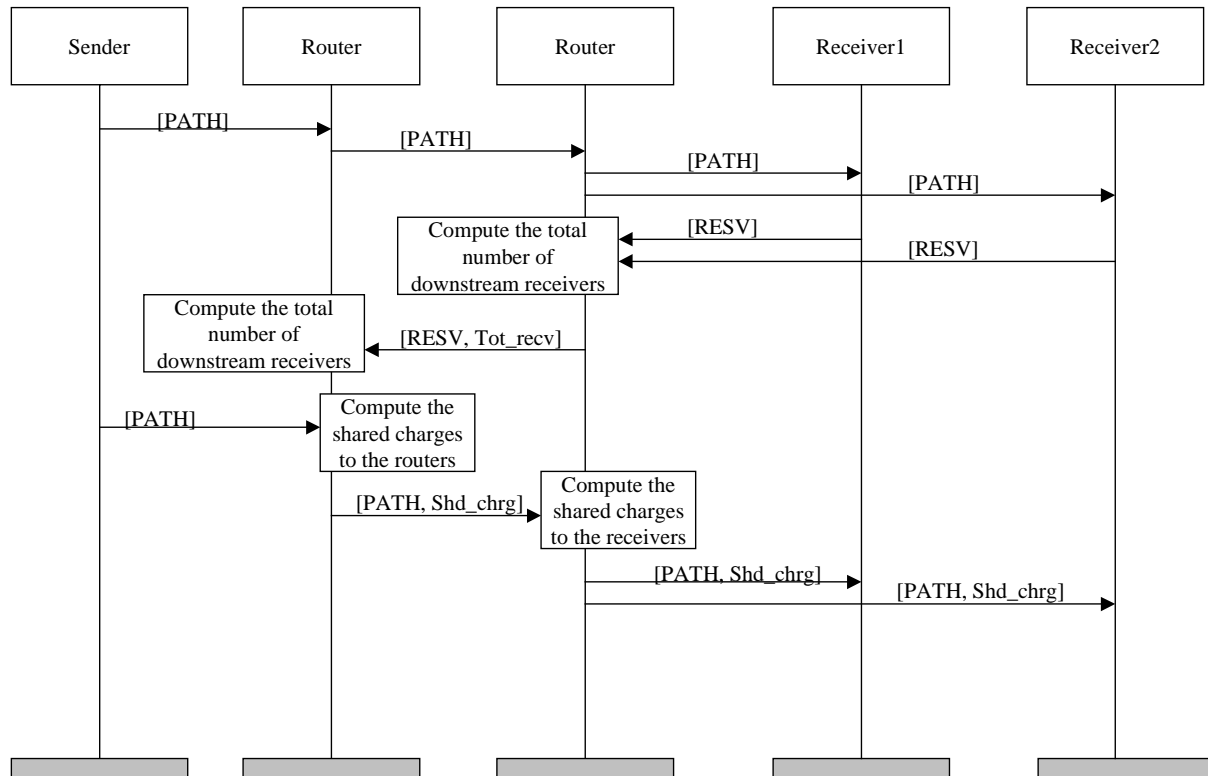


Figure 8: MSC for CAP in receiver-paying model

In the receiver-paying model, Shai Herzog's algorithm [Herz96a] discussed in section 3.3 is used to shared the cost among the receivers when they have different QoS levels on a common link.

In the receiver-paying model, an RSVP node periodically receives the „downstream receivers“ elements contained in Resv messages from all of its next downstream hops. It computes the total number of its downstream receivers, creates a new „downstream receivers“ element and sends this element encapsulated in a Resv message to its next upstream hop. This RSVP node, in turn, computes the total number of its downstream receivers and reports this information to its next upstream hop. By using this mechanism, an RSVP node can obtain the total number of its downstream receivers. In turn, it has to tell its next upstream hop about this so that the upstream hop can also compute the total number of its downstream receivers.

When an RSVP router receives an „upstream charge“ element contained in a Path message from its next upstream hop, it knows the charge of the multicast subtree, which spreads from the sender to itself, to all of its downstream receivers. In the ELSD scheme, all receivers downstream of a multicast subtree have to share the cost of that multicast subtree equally. The idea of our solution in the receiver-paying model is depicted in figure 9.

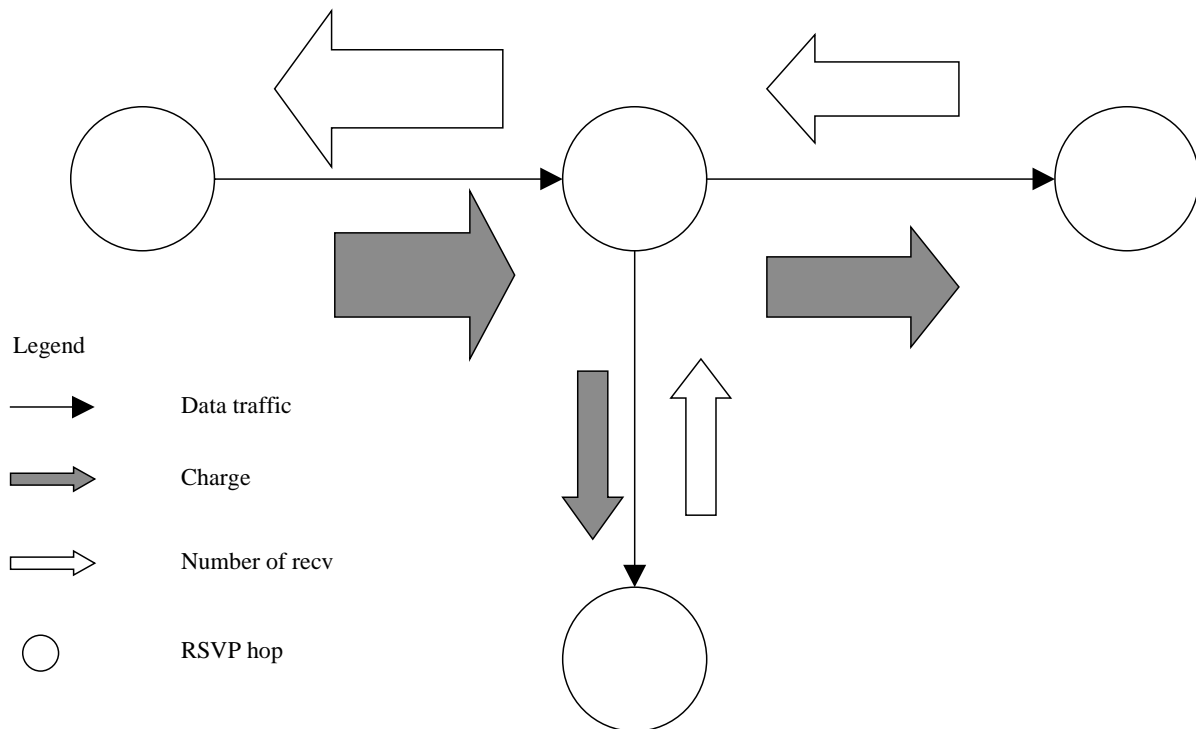


Figure 9: Exchange of charging and accounting information in receiver-paying model

The router computes the shared charge to a single downstream receiver by dividing the „upstream charge“ by the total number of its downstream receivers (the router can know the total number of its downstream receivers by learning that from its next downstream hops as described above). It then computes the total charges to each of its next downstream hop by multiplying the shared charge to a single downstream receiver by the total number of downstream receivers of that next downstream hop.

The RSVP node creates the new „upstream charge“ elements for each of its next downstream hops by adding the cost of the downstream link to the shared charges to that node and sends these elements encapsulated in Path messages to its next downstream hops. The shared charge to a receiver is the sum of shared cost of the upstream multicast subtree and the shared cost of the local network link cost. In the ELSD scheme, the local network link cost is shared equally by all receivers on that network and all downstream receivers whose upstream routers receive the traffic on that local network.

Because CAP elements are encapsulated in RSVP messages and sent over IP with „best effort“ transport service between the upstream and downstream nodes, CAP must use a mechanism to enable a reliable communication between them. The mechanism works as follows:

Each “upstream charge” element is associated with a sequence number. When the CAP module receives an “upstream charge” element from its upstream neighbor, it has to check whether the CAP element has arrived correctly and in order or not. If the CAP element is correctly received and its sequence number is equal to the sequence number that the CAP module expects to receive, the CAP module will proceed to compute the shared charges and send them downstream to its downstream neighbors as described above. The CAP module increments the expected sequence number and puts this number into the “downstream

receivers” element to notify its upstream neighbor that the last “upstream charge” element has been correctly received.

A CAP module maintains a queue for each of its downstream neighbors that contains the “upstream charge” elements which have been sent downstream but have not yet been acknowledged. When the RSVP module sends a Path message to the downstream neighbor, the CAP module encapsulates the “upstream charge” element inside the Path message and has the RSVP module send them downstream. When the CAP module receives an acknowledgement for its “upstream charge” element, it can be sure that the element has been received and can thus dequeue that “upstream charge” element from its queue.

In theory, the maximal number of “upstream charge” elements that can stand out for their acknowledgement is L where L is the time after which the RSVP states time out. The current standard of RSVP [Brad97] specifies that L must fulfill the requirement $L \geq (K+0.5)*1.5*R$ where R is the refresh period of RSVP messages and K is chosen in a way that $K-1$ successive messages can be lost without the RSVP states being timed-out¹.

The value of R is contained in a TIME_VALUES object in a Path or Resv message and specifies the refresh period used by the RSVP hop to send the Path or Resv message. Thus R and L may vary from hop to hop. If an RSVP hop chooses a small value of R while its downstream hop chooses a large one, the number of “upstream charge” elements sent is higher than the number of acknowledgement received (however, an acknowledgement can acknowledge multiple “upstream charge” elements). This can result in a large queue and is critical when a Resv message (and thus the acknowledgement encapsulated) is lost.

A solution for this problem is proposed as follows:

At a moment, a CAP module may have maximal N entries containing the “upstream charge” elements for any of its downstream neighbors and the maximal number of “upstream charge” elements standing out for acknowledgement is $N-1$, i.e. the CAP module may send maximal $N-1$ consecutive “upstream charge” elements to any of its downstream neighbors without receiving any acknowledgements for them. When the number of transmitted “upstream charge” elements hits $N-1$ and there is a new charge to the downstream neighbor, the CAP module generates the N -th entry in the queue but does not send an “upstream charge” downstream. Every time when there is a new charge to the downstream neighbor, the CAP module adds this new charge to the charge contained in the N -th entry in the queue, i.e. the N -th entry is used to accumulate the new charges to a downstream while waiting for an acknowledgement.

The CAP module continues to do so until the downstream neighbor is timed-out or an acknowledgement encapsulated in a Resv message is received. In the first case, the CAP module clears the queue and writes the charging information in a log file. In the second case, the CAP module removes the acknowledged “upstream charge” entries from the queue. Now, the number of “upstream charge” standing out for acknowledgement is lower than $N-1$; the (former) N -th “upstream charge” entry is treated as a normal entry and can be transmitted downstream encapsulated in the next Path message.

The reason why we do not allow the CAP module to send the N -th “upstream charge” downstream is argued as follows:

¹ In RSVP, the default values for K and R are 3 and 30 seconds, respectively.

If there were a new charge to the downstream neighbor after the N-th “upstream charge” element had been sent downstream and the CAP module added this new charge to the N-th “upstream charge” entry and sent it downstream again, the CAP module would not be able to tell which of the N-th “upstream charge” were acknowledged when an acknowledgement for the N-th “upstream charge” element were received.

In order to improve the reliability of the charging and accounting protocol, we also suggest that a CAP module have a log file for each of its downstream receivers. The charges to a downstream node should be written in this log file besides being sent downstream. This log file is allowed for the case that the computer or router hosting a CAP module suddenly crashes, resulting in losing the charging information. The log file can also be used to create the bill to a user (e.g., at the end of the month).

Sender-paying Model

The message sequence chart in figure 10 shows the flow of the RSVP messages and CAP objects in the sender-paying model.

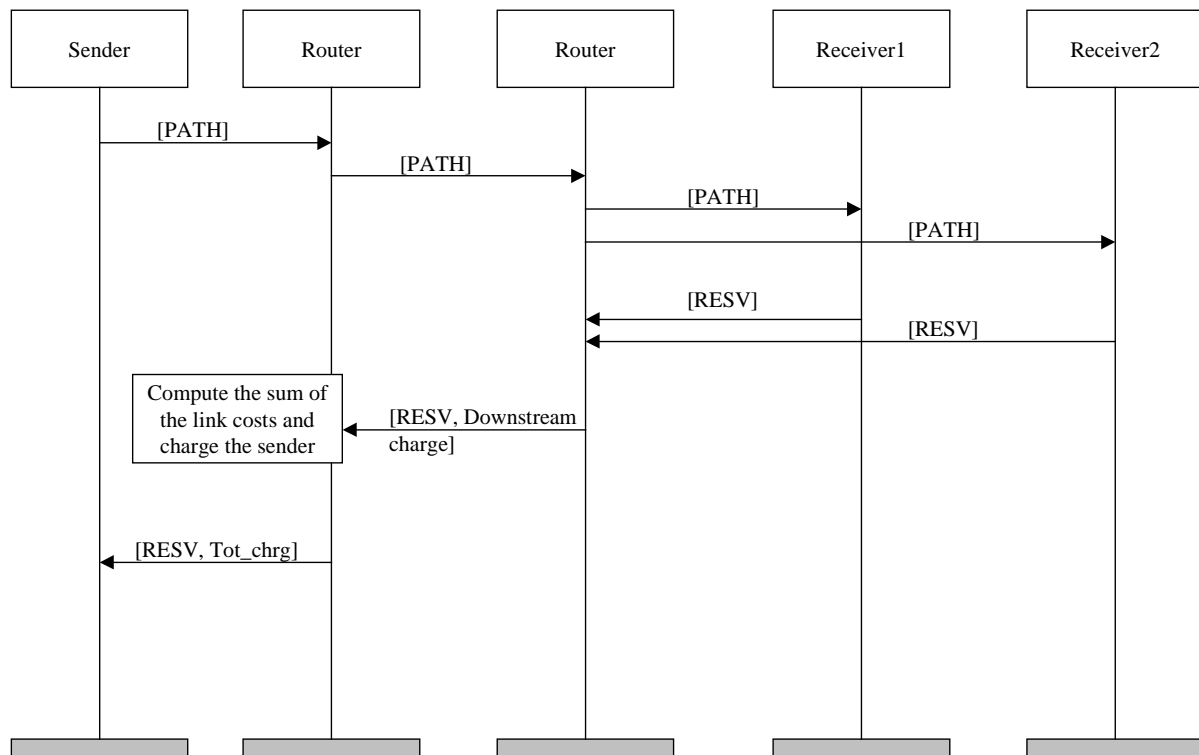


Figure 10: MSC for CAP in sender-paying model

As already mentioned above, we simply assume in this section that a router can somehow determine the cost of its downstream links. Refer to Chapter 3 for more details. In the sender-paying model, an RSVP node periodically sends its „downstream charge“ element embedded in Resv messages to its next upstream hop. The „downstream charge“ of an RSVP node is the cost of the multicast subtree downstream of that RSVP node. If an RSVP node is a receiver, then its „downstream charge“ is zero because there is no multicast subtree downstream of it.

After having received the „downstream charge“ elements from all of its next downstream hops, the RSVP node then computes the sum of its downstream charges and adds the costs of the links on which the „downstream charge“ were received to that sum to create its „downstream charge“ element. RSVP encapsulates its „downstream charge“ element in a Resv message and sends it to the next upstream hop to inform the next upstream hop about the cost of its downstream multicast subtree. The flow of “upstream charge” elements continue to propagate upstream until it hits the sender where the charging finally takes place. The idea of our solution in the sender-paying model is depicted in figure 11.

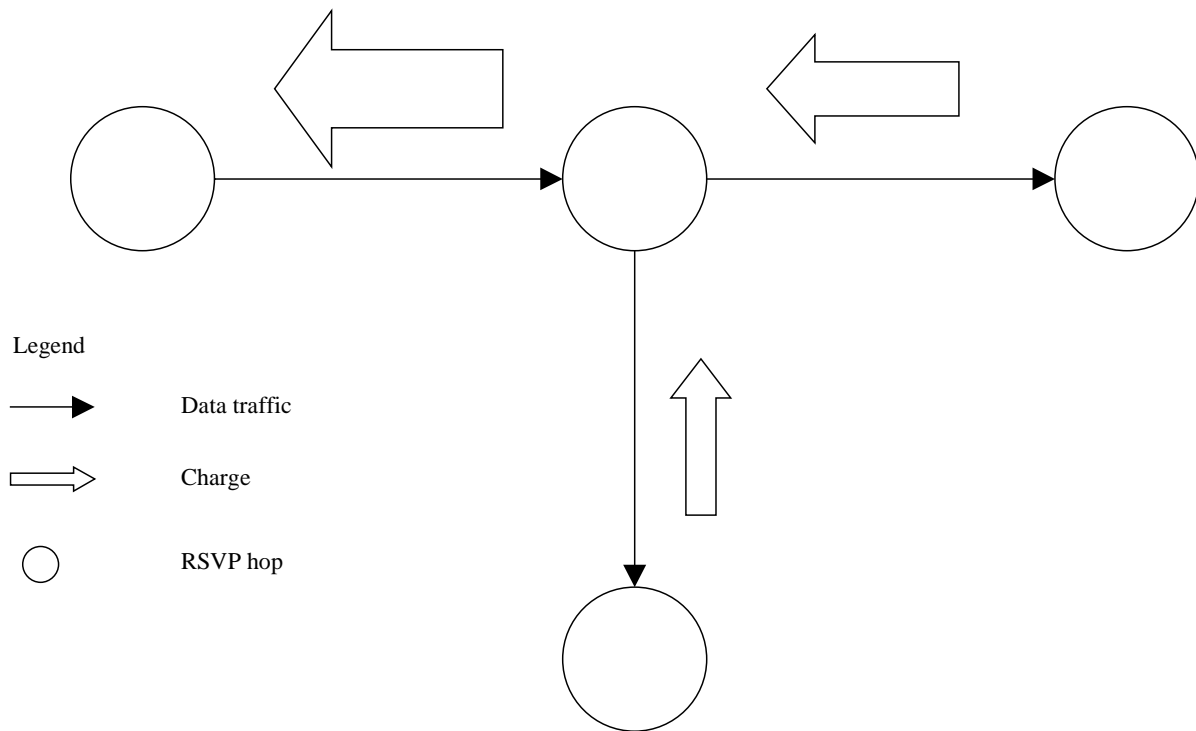


Figure 11: Exchange of charging and accounting information in sender-paying model

Analogous to the receiver-paying model, the sender-paying model also has a mechanism enabling a reliable communication of the charging information between the CAP modules. The mechanism works as follows:

Each “downstream charge” element is associated with a sequence number. When the CAP module receives a “downstream charge” element from its downstream neighbor, it has to check whether the CAP element has arrived correctly and in order or not.

If the CAP module receives a “downstream charge” element from its next downstream neighbor that contains a sequence number different from that it expects, it simply discards that “downstream charge” element. If the “downstream charge” element contains a sequence number equal to the expected sequence number, the CAP module reads the charging information from that CAP element and charges it to its upstream neighbor. In this case, the expected sequence number in the CAP module is incremented. Next time, when the CAP module is prompted by RSVP for a downstream outgoing CAP element, it builds a “downstream charge acknowledgement” element containing the sequence number of the “downstream charge” element from the downstream neighbor it has correctly received. Thus, the sequence number contained in the “downstream charge acknowledgement” element transferred downstream is the acknowledgement for the last correctly received charging

information in the upstream direction. When the “downstream charge acknowledgement” element arrives at the downstream hop, it reads the sequence number contained in the element and can be sure that the charging information has been received by its upstream hop.

Analogous to the receiver-paying model, the CAP modules in the sender-paying model also maintain a list of “downstream charge” elements that have been sent to the upstream neighbor but have not yet been acknowledged. When the CAP module is prompted by the RSVP module for upstream outgoing CAP objects, it encapsulates the “downstream charge” elements inside the Resv message and has the RSVP module send them upstream. When the CAP module receives an acknowledgement for its “downstream charge” element, it can be sure that the element has been received and can thus dequeue that “downstream charge” element from its queue.

In the sender-paying model, the CAP module also applies the same method to limit the size of the queue containing “downstream charge” elements as in the receiver-paying model and has a log file for its direct upstream neighbor to allow for crashes resulting in losing the charging information.

6. Summary and Future Work

6.1 Summary

In this work, we have discussed different problems of charging and accounting for the current Internet. We have presented some service scenarios that we consider typical for the types of the new applications. We discussed the general issues of charging and accounting in computer networks, in particular the special issues of this topic in ATM and IP networks and the general requirements a charging and accounting protocol should fulfill. We then presented the potential solutions of a charging and accounting protocol in the MIS architecture and discussed their advantages and disadvantages. Finally, we extended our solution to have a general charging and accounting protocol. We then concluded this work with some implementation details of our prototype implementation.

6.2 Future work

The following issues are left unexplored in our works and will be tackled in our future work:

Charging and accounting with shared reservation filter style: In our prototype implementation, we only support the service scenarios with one sender. Thus, our charging and accounting protocol only supports fixed-filter style reservation. Shared reservation (i.e., shared-explicit filter and wildcard filter style reservation) is a challenging topic that should be targeted in our future work.

Details of computing the costs of a link: In this work, we contended that the cost of a connection, either unicast or multicast, must be computed based on the reservation and the consumption of the network resources caused by that connection. However, we did not delve into details of how to compute the cost of a connection and simply used a simple formula to compute the cost of a single link based on its reservation parameters of the Rspec [Brad97, Wroc97]:

$$\text{Cost} = \text{Token Bucket Rate} * \text{Token Bucket Size}$$

In our prototype implementation, this formula is applied to the both integrated service models Controlled-Load and Guaranteed QoS control services. In parallel with this work, we are also investigating the methods to compute the cost so that this simple formula will be supplanted by a more meaningful formula.

Optimal time intervals for computing costs and information exchange: A challenging issue left unexplored in this work is about the time interval at which the costs are computed and shared among the users and the time interval of information exchange between neighbor CAP nodes. In our prototype implementation, we let these time intervals be equal to the refresh period of RSVP messages¹. In future work, we will try to find the optimal constant values for these time intervals or a meaningful mechanism that adapts these time intervals depending upon the network conditions. More research and experiments are needed to do this.

¹ The refresh period of RSVP messages is 30 seconds by default.

Appendix A RSVP-CAP Interface

The RSVP-CAP interface is based on the API styles defined in [Herz96a, Herz96b, Rap98a]. Note: [Rap98a] is the currently published specification of the interface between RSVP and policy control in the IETF working group RAP. However, we slightly changed the interface to adapt it to our protocol. In this section, we will provide the original interface of [Rap98a], our modified interface for RSVP-CAP and the arguments for the modification.

A.1 The RSVP-LPM Interface [Rap98a]

- Common parameters:

The following is a list of common parameters used by several policy control functions. The terminology used here is partly borrowed from RSVP. Refer to [Brad97] for more details.

Session

An RSVP session defines one simplex unicast or multicast data flow for which reservations are made. A session is identified by the destination address, the transport layer protocol and an optional (generalized) destination port.

Filter_spec_list

The filter_spec_list defines the set of flows to which the POLICY_DATA objects apply.

Shr_ind

Shr_ind indicates whether the reservations are shared.

Rsvp_hop

The parameter rsvp_hop specifies the neighbor policy node (upstream or downstream), including its local LIH (logical interface handle [Brad97]).

Message_type

The parameter message_type specifies the type of message that carries the POLICY_DATA objects. Note: The type of message implies the direction of the message (upstream or downstream).

Resv_flowspec

The resv_flowspec contains information on the current or desired reservation and traffic characteristics.

Resv_handle

The resv_handle specifies a reservation channel within a session.

Cbp and giveup_time

Cbp is the address of a callback function and giveup_time is the maximal time RSVP is willing to wait. The value of cbp is echoed back in the function upcall notifying the results. If giveup_time expires, upcall returns an error code.

In [Rap98a], the following API calls are provided by the policy control module to RSVP:

- `PC_InPolicy` (`message_type`, `rsvp_hop`, `session`, `shr_ind`, `filter_spec_list`, `in_policy_objects`, `resv_handle`, `resv_flowspec`, `refresh_period`, `cbp`, `giveup_time`) → return code.

This function is called when the RSVP receives an incoming message containing `POLICY_DATA` objects.

- `PC_Outpolicy` (`message_type`, `rsvp_hop_list`, `session`, `shr_ind`, `filter_spec_list`, `max_pd`, `avail_pd`, `cbp`, `giveup_time`, `out_policy_objects`) → return code.

RSVP calls this function to get outgoing `POLICY_DATA` objects for its outgoing messages. `Max_pd` and `avail_pd` specify the desired maximal object size and the available space within the RSVP message.

- `PC_AuthCheck` (`message_type`, `session`, `shr_ind`, `filter_spec_list`, `resv_desc_list`, `full_list_ind`, `cbp`, `giveup_time`) → return code.

RSVP calls this function to check the authorization of the reservation. The `resv_desc_list` is a list of reservation descriptions. Each reservation description is identified by the LIH, the `resv_handle` and the `resv_flowspec`. Because there may be multiple downstream LIHs making the same reservations, authorization can be checked separately for each LIH, once for all LIHs, or anything in between. `Full_list_ind` is used as an indication for the last authorization check of the series. This feature is useful when the cost is shared among the downstream receivers because the server needs to know the exact number of the receivers in this case.

- `PC_Init` (`K`, `upcall`, ...) → return code.

This call initializes the LPM and sets some RSVP/policy configuration parameters. `K` is the “soft state” multiplier for refresh period and `upcall` the address of a function that is called by the LPM when policy changes.

- `upcall` (`event_type`, `cbp`, `message_type`, `lih`, `rsvp_hop_list`, `session`, `shr_ind`, `filter_spec_list`, `out_policy_objects`, return code).

This function is called by the LPM to inform the RSVP about policy changes. `Event_type` specifies the type of the original call. `Cbp` has the same value as the `cbp` provided by the RSVP when making the original call.

- `PC_DelState` (`message_type`, `rsvp_hop`, `session`, `filter_spec_list`, `op_type`) → return code.

This call is made when the route or other RSVP states are changed by RSVP. It affects the policy state corresponding to the RSVP state. `Op_type` specifies the type of the change request:

`PC_Delete`: Delete the policy state.

`PC_Block`: Block (ignore) the state.

`PC_Unblock`: Unblock the state.

A.2 The RSVP-CAP Interface

[RAP98a] proposes the asynchronous and synchronous model for the API calls. In the synchronous model, the calls are blocking functions (i.e., RSVP is blocked until the calls are completely done and the results are returned). In the asynchronous model, the calls are non-blocking functions (i.e., they do not block RSVP) and their delayed results are returned to

RSVP later through an upcall by the LPM. In the asynchronous model, RSVP must suspend the incomplete tasks, save their contexts and complete them later when the results of the calls are returned. The implementation of the asynchronous model is more difficult but it can gain better scaling properties when the LPM has to consult a remote policy server with higher authority in a client-server paradigm.

Our protocol only supports the synchronous model because it does not deal with policy control but with charging and accounting. In the context of our protocol, a CAP module communicates the charging and accounting information piggybacked in RSVP messages with its neighbor CAP modules in a peer-to-peer paradigm. Thus, the CAP never has to block the calls to consult a remote server¹. In our protocol, RSVP calls the API functions when it receives an RSVP message containing CAP objects or when it is about to send an RSVP message to its neighbors. The CAP processes the incoming CAP objects or creates the outgoing CAP objects and returns the results to RSVP. Thus, it does not make sense to support the asynchronous model in this case because it would approximately have the same scaling and performance properties as the synchronous model.

- Common parameters:

Flow_handle

The flow_handle corresponds to a simplex unicast or multicast data flow with a desired reservation level and traffic characteristics. It replaces the two parameters session and resv_handle.

Vif

The term Virtual Interface (vif) is borrowed from DVMRP[Wait88]. It specifies the interface on which the CAP objects are received or sent embedded in RSVP messages. Vif helps the CAP module determine the links connecting itself to its neighbors and build the topology of the multicast tree. Note that the CAP module needs to know on which links it receives or sends the CAP objects from or to its CAP neighbors in order to determine the cost of the links. The virtual interface is only of importance and differs from the physical interface when there is a network which does not support multicast between a router and its next downstream router. In this case, each IP multicast datagram is encapsulated in conventional IP datagram and sent to the next downstream router. This technique is called tunneling. Thus, the link cost of the virtual connection must be the sum of the cost of the physical links.

Pcapobj

Pcapobj is a pointer to CAP objects.

Mtype

The parameter mtype specifies the type of the message carrying the CAP objects. Note: The type of the message implies the direction of the message (upstream or downstream).

Rsvp_hop

The rsvp_hop specifies the neighbor RSVP node (upstream or downstream), including its local LIH (logical interface handle [Brad97]).

¹ Although our protocol has a server or maybe more, the CAP server(s) are only in charge of computing the charges and debiting the hosts participating in multicast sessions. The communication between a CAP server and the CAP modules takes place periodically and does not block the API calls.

Qos

Qos specifies the QoS level of the flow. In the MIS architecture, the desired reservation and traffic characteristics are quantized, and receivers with approximately the same desired reservation are grouped to share a point-to-multipoint VC with a quantized QoS level¹. The QoS level reflects the reservation and traffic characteristics of the flow.

The following parameters are not used in our interface:

- The parameters `shr_ind` and `filter_spec_list` in the original RSVP-LPM interface are not needed in our API calls because the case of multiple senders (i.e., multipoint to multipoint) is not yet supported in the MIS architecture. Thus, our prototype implementation only supports reservations with wildcard style.
- The parameter `resv_flowspec` in the original interface is replaced by the parameter `qos` in our RSVP-CAP interface.
- The parameters `cpb` and `giveup_time` in the original RSVP-CAP interface are not needed in our API calls because our implementation only supports the synchronous model and does not support the asynchronous model.

The following calls of the original RSVP-CAP interface are not supported by our implementation:

- The call `PC_AuthCheck` is not supported because our protocol does not deal with policy control.
- The call `upcall` because our implementation only supports the synchronous model.

The following calls are provided by our CAP module to RSVP via the RSVP-CAP interface:

- `cap_open (vif, rsvp_hop, qos)`

This call is made when RSVP receives the first Path message from its upstream neighbor and establishes the Path state. The call returns a flow handle which is opaque to RSVP. RSVP uses this flow handle to refer to the flow when it makes the API calls to the CAP later.

In RSVP code, it is called in the procedure `accept_path` contained in the file `rsvp_path.c`.

- `cap_in_upstr(flow_handle, vif, pcapobj, rsvp_hop) → return code.`

This call corresponds to the call `PC_InPolicy` in A.1 for CAP objects encapsulated in Path messages.

In RSVP code, it is called in the procedure `accept_path` contained in the file `rsvp_path.c`.

- `cap_in_downstr(flow_handle, vif, pcapobj, mtype, rsvp_hop, qos) → return code.`

This call corresponds to the call `PC_InPolicy` in A.1 for CAP objects encapsulated in Resv messages.

¹ This technique is called granular QoS support.

In RSVP code, it is called in the procedure `accept_resv` contained in the file `rsvp_resv.c`.

- `cap_out(flow_handle, vif, mtype, rsvp_hop) → pcapobj`.

This call corresponds to the call `PC_OutPolicy` in A.1.

In RSVP code, it is called in the procedure `path_common_refresh` contained in the file `rsvp_path.c` and in the procedure `resv_refresh_PSB` contained in the file `rsvp_resv.c`.

- `cap_prune(flow_handle, vif, mtype, rsvp_hop, qos) → return code`.

This call corresponds to the call `PC_DelState` in A.1. However, it is only used to delete the states in the CAP module. RSVP makes this call to CAP when it receives a `PathTear` or a `ResvTear` message, or when the `Path` or `Resv` state times out.

In RSVP code, it is called in the procedure `kill_RSB1` contained in the file `rsvp_resv.c` and in `kill_session` contained in the file `rsvp_util.c`

- `cap_config() → return code`.

This call corresponds to the call `PC_Init` in A.1.

In RSVP code, it is called in the main procedure contained in the file `rsvp_main.c`.

Appendix B Implementation Details

In this section, we provide the readers with the details of the prototype implementation of our charging and accounting protocol specified in the previous chapters. The implementation and testing of our protocol took several months to carry out and the code consists of more than 1,000 lines. Code is available upon request.

Our prototype implementation provides an API function library to the RSVP as described in Appendix A. Although it was possible to extend the MARPTable of the EARTH server or the State Tables of the RSVP module to contain the charging and accounting information, it was decided to implement the CAP module with a separate table that contains the charging and accounting information. This is an important decision that makes the CAP independent on the network technologies and reservation signaling protocols. Thus, the CAP can run on any network technologies in conjunction with any reservation signaling protocols similar to RSVP. The only requirement which must be fulfilled is that the CAP modules do not have a mechanism to transport information themselves and have to use a transport protocol¹ to exchange the charging and accounting information with their neighbor nodes.

At the moment, our charging and accounting protocol only supports the receiver-paying model because this model is of our major interest in extending the MIS architecture with the capability of charging and accounting. In future work, we will extend CAP to support the sender-paying model. We believe that this work is fairly straight forward when exploiting the experiences we have gained during implementing the receiver-paying model.

Figure 12 illustrates the functionality of a CAP module and its table containing the charging and accounting information. As shown in figure 12, the router R2 is connected to its upstream router via the interface Vif1. It has two downstream routers connected to it via the interface Vif2 and Vif5. Router R2 has two receivers directly receiving the multicast data from it via the interface Vif3 and Vif4.

The data structure of the CAP module located in router R2 shown in figure 12 is slightly different from that of our prototype implementation. It is simplified in order to help the readers quickly get an understanding of the functionality of the CAP modules.

The data structure of the CAP module located in router R2 shown in figure 12 contains information about the total number of its downstream receivers, the QoS level it has reserved for them, and a record containing information about its upstream hop, such as the IP address of the upstream hop and the interface connecting to it. It also has a list of records containing information about its next downstream hops, such as their IP address, the interface connecting to them, the total number of their downstream receivers, the QoS level they have reserved for their downstream receivers and the charge to them.

The charge to a downstream hop is the sum of the shared charge to that downstream hop and the cost of the link connecting to it. The shared charge to a downstream hop is computed according to the ELSD as discussed in chapter 3, 4, and 5. The downstream hops of R2 must always inform it about the total number of their downstream receivers by encapsulating this

¹ When running in conjunction with RSVP, CAP encapsulates its charging and accounting information in RSVP messages (Path and Resv) and uses RSVP as a transport protocol.

information in Resv messages and sending it upstream to R2¹. When R2 receives the charge from its upstream hop R1, it applies the ELSD scheme to compute the shared charges to its downstream hops, adds the link costs to these shared charges and sends them downstream.

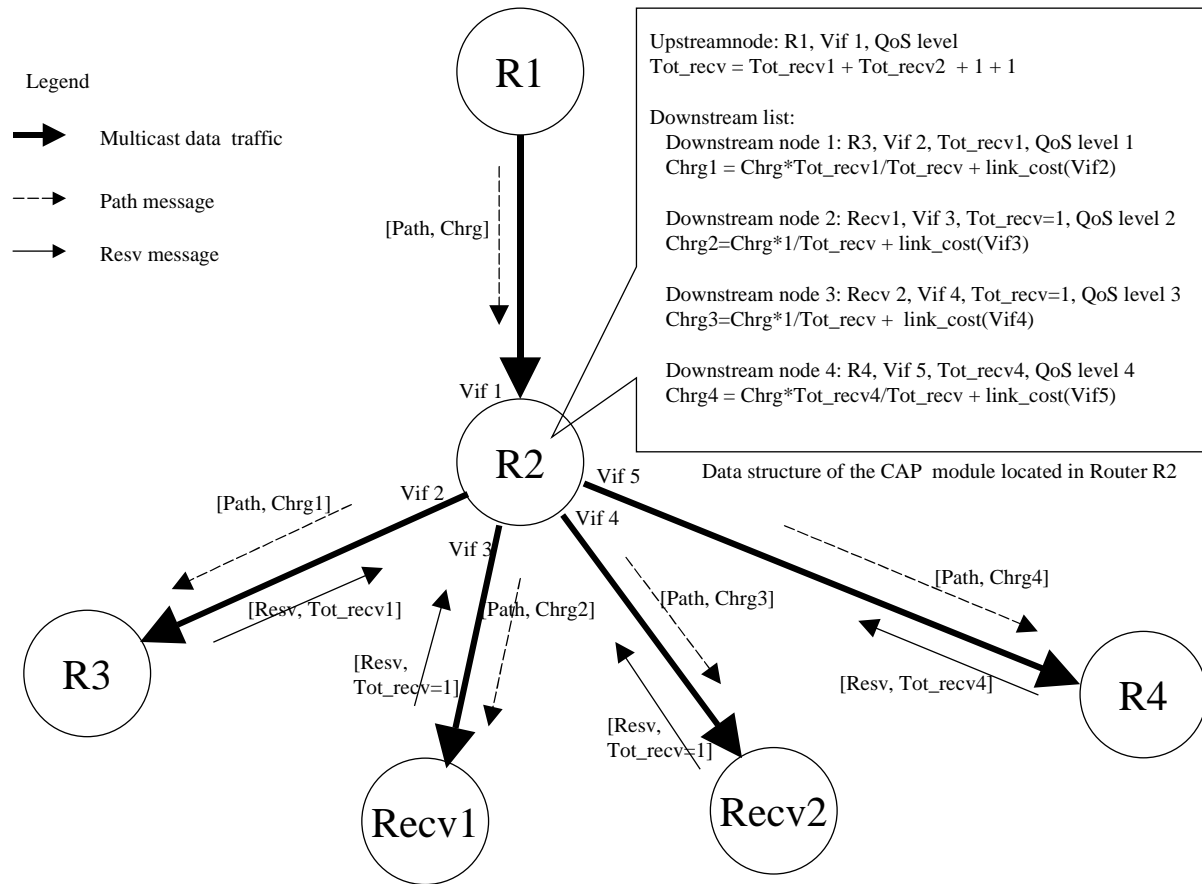


Figure 12: Functionality and data structure of a CAP module

In our prototype implementation, the CAP module has a dynamic linked list of such data structures. However, they are slightly more complicated than the data structure presented above. Each data structure contains the charging and accounting information for a flow pertaining to a session², such as the QoS level it has reserved for its downstream receivers and information about its next upstream and downstream hops.

The data structure has a record containing information about its next upstream hop, such as the IP address of the next upstream hop, the interface connecting to it and the sequence number of the last received objects containing charging information. The sequence number enables the CAP modules to communicate the charging information reliably. The CAP modules use the “Go Back n” protocol to communicate the charging information with its next downstream hops as described in chapter 5. The data structure of a CAP module is shown in figure 13.

¹ The total number of downstream receivers of Recv1 and Recv2 is one because they are end system (i.e. host).

² In RSVP, a session is defined by the destination address (e.g., IP unicast or multicast address), the transport layer protocol (e.g., UDP), and an optional destination port.

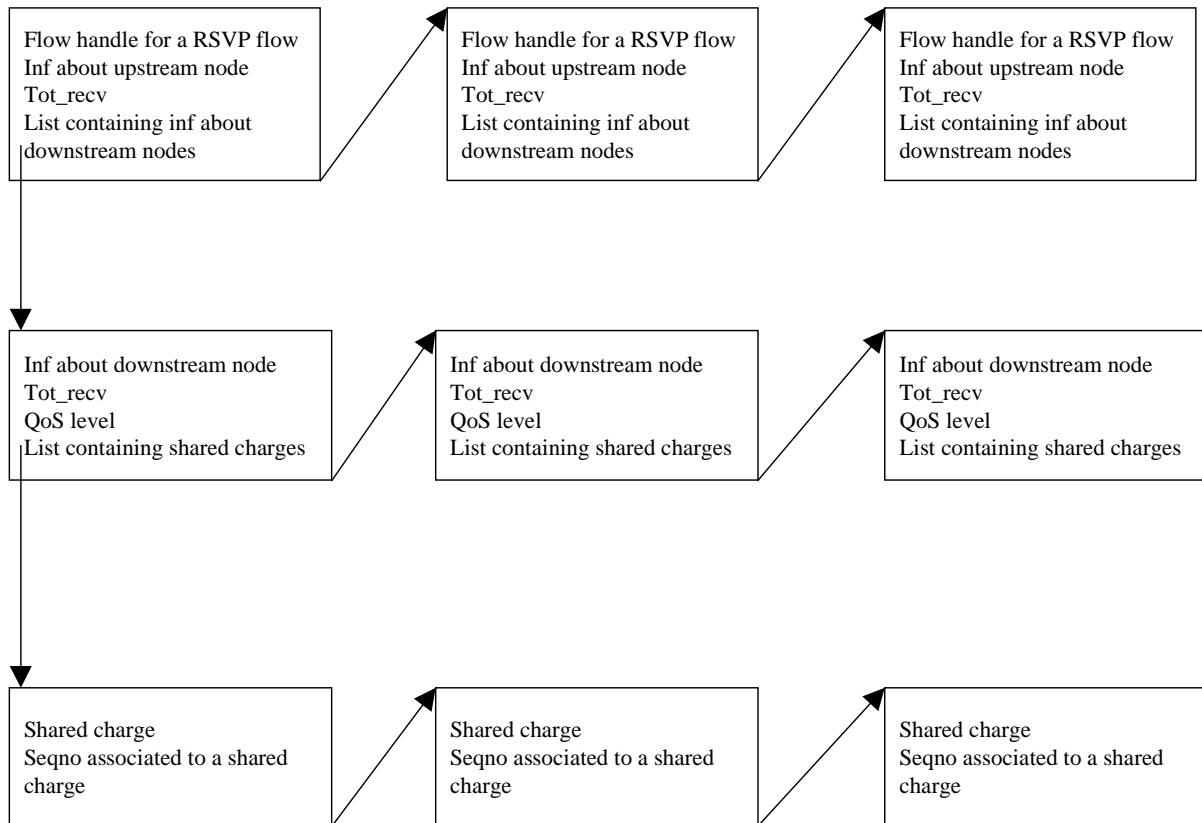


Figure 13: Data structure of a CAP module containing charging and accounting information

Each data structure of a CAP module has a dynamic linked list of records containing information about its next downstream hops, such as their unicast IP address, the total number of their downstream receivers, their desired and reserved QoS level and a linked list of CAP object elements containing charging information which have been sent downstream but have not yet acknowledged by the downstream hops. When the CAP module receives a CAP object with a sequence number N from its downstream hop, it removes the records containing charging information with sequence number smaller than N from the linked list because they all have been correctly received and acknowledged by the downstream hop.

Another problem we did not mention in the specification of the charging and accounting protocol is that the RSVP application is also considered as a “RSVP hop” in the context of the RSVP software. Thus, the application and the RSVP located on the same host also exchange RSVP messages as if they were upstream and downstream neighbors. RSVP keeps application from being timed-out by creating Path or Resv messages (depending on whether the application is sending or receiving data) for the application and then sending these messages to itself.

To adapt our implementation to this feature of RSVP, we additionally define two more CAP elements:

- `api_cap`:

This CAP element is contained in RSVP messages passed on from the RSVP application to RSVP. It does not have any specific data and simply notifies that it is sent from an RSVP application.

Just like in the context of RSVP, an RSVP application does not build itself this CAP element. Instead, RSVP makes an API call to the CAP module to get this CAP element. This circumstance results in two API calls to the function `build_api_cap_pkt` in the procedures `api_prepare_path` and `api_prepare_resv` contained in the file `rsvp_api.c`

- `cap_api`:

This CAP element is contained in RSVP messages passed on from the RSVP to RSVP application. Currently, it does not have any specific data and simply notifies that it is sent from RSVP to an RSVP application. However, we may modify this CAP element to notify the users of more charging and accounting information (e.g., currently accumulative charge to the user or expected shared charge in the next time interval).

The functionality and the state transition diagram of a CAP module are depicted in figure 14 to help the readers gain a quick understanding of our charging and accounting protocol's functionality in receiver-paying model.

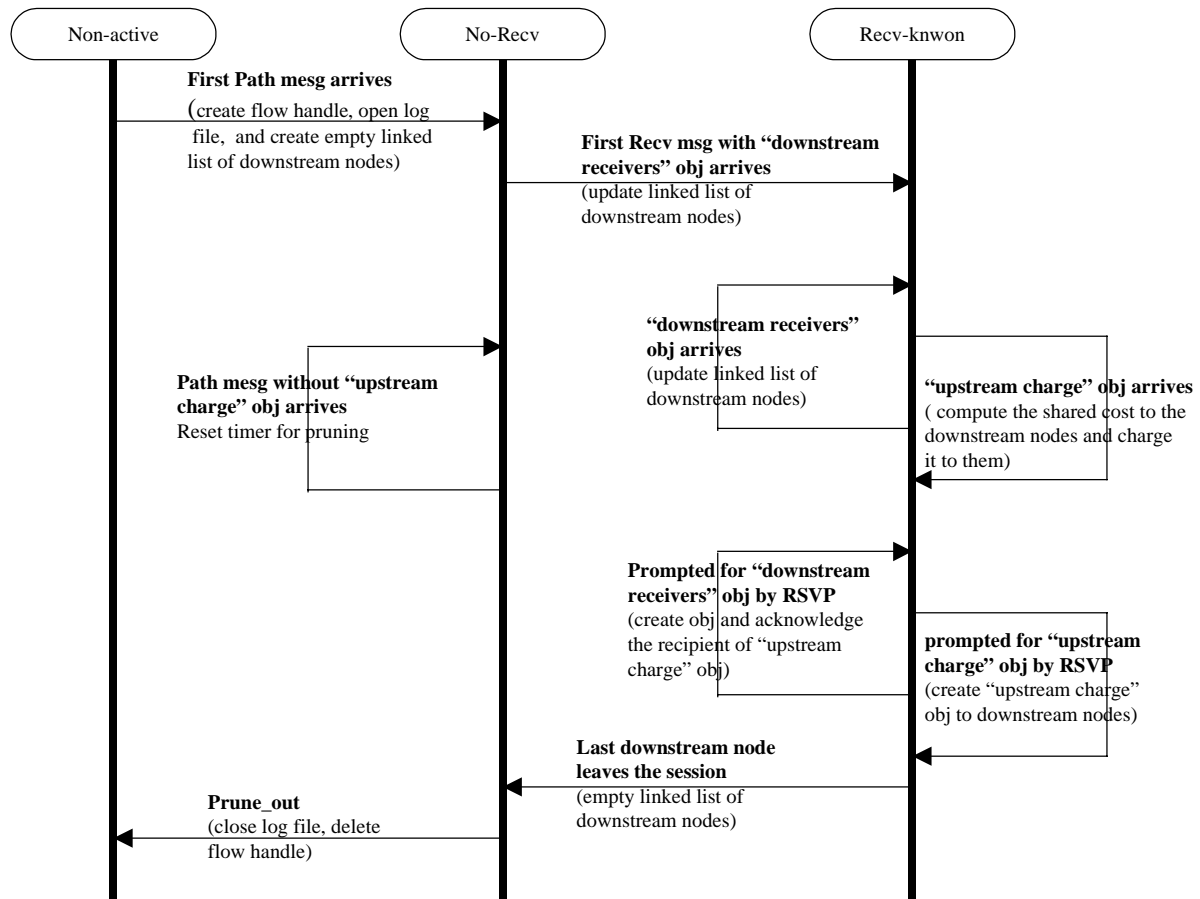


Figure 14: State transition diagram of a CAP module in receiver-paying model

Appendix C Testing of the Implementation

This appendix provides the interested readers with a proof of our charging and accounting protocol's functionality. It is divided into two sections. The first section demonstrates the functionality of the protocol in a "simulated RSVP environment". The second section demonstrates the functionality of the protocol in the real RSVP environment.

The "simulated RSVP environment" is a testing program that invokes the API calls to CAP and checks whether the CAP module reacts to these calls as specified by the protocol. This simulated environment helped to separate CAP from RSVP and to locate and fix bugs at the beginning of the implementation. Moreover, this environment enables us to test and observe some special behavior of the CAP which is very cumbersome or even impossible in the real environment RSVP (e.g., message lost, message delivered in wrong order etc.).

C.1 Testing of the Implementation in the simulated RSVP environment

This section demonstrates the functionality of a CAP module located inside a router in a "simulated RSVP environment". The figure below depicts a typical message exchange sequence of the CAP module (193.175.34.9) and its up- and downstream hops. Note that this message sequence can change depending upon the difference of the CAP modules' refresh periods and how often the CAP objects are lost. Figure 15 has a scenario as follows: The router "193.175.34.9" has an upstream hop with the address "193.175.33.9" and three downstream hops with the address "193.175.35.9", "193.175.36.9", "193.175.37.9". The router "193.175.34.9" receives the upstream charge from its upstream hop and shares this charge among the three downstream hops based on their QoS levels and their total number of downstream receivers.

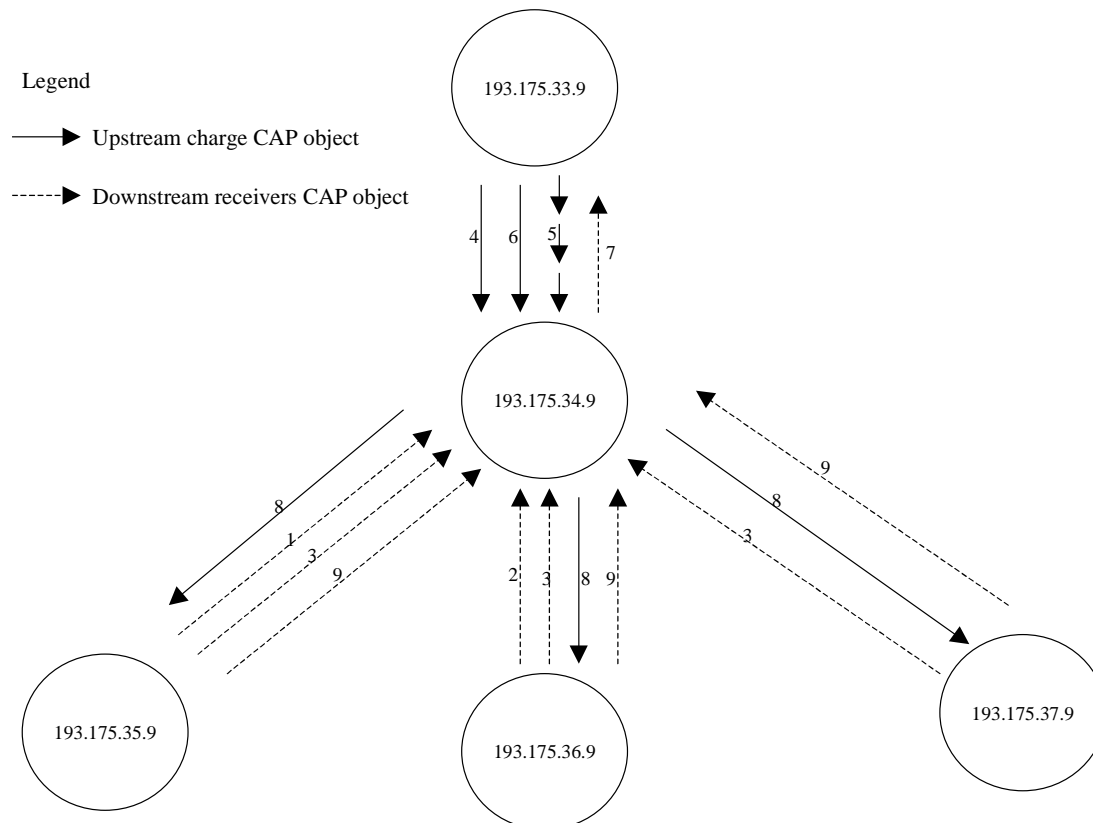


Figure 15: Message exchange and functionality of a CAP module located inside a router

Following, the functionality of the CAP module located inside the router “193.175.34.9” is demonstrated with the help of a log file written during the multicast session.

Typographical conventions: In this appendix, the message dump is printed in **bold font** and the dump of the charging and accounting information inside the CAP module is printed in *italic font*.

1. Step 1: The CAP module receives the first “downstream receivers” CAP element contained in a CAP object from a downstream hop.

There is only a Path state in place but no Resv states have been established yet.

The list containing charging and accounting information inside the CAP module before the CAP object arrives:

flow handle = 0 ← flow handle pertaining to an RSVP flow.
tot_recv = 0 ← Total number of receivers.
Upstream...193.175.33.9 ← IP address of the upstream hop.
Upstream seqno=0 ← „Go Back n“ sequence number of „upstream charge“ CAP elements.
Upstream vif = 1 ← Interface connecting to the upstream hop.
Downstream node list ... (None) ← Dynamic linked list containing information about downstream hops.

The CAP module receives the first „downstream receivers“ CAP element contained in a CAP object passed on by RSVP module:

Length: 32 ← Total length of the CAP object.
Class: 14 ← Policy class (specified by RSVP).
Type: 1 ← Policy type (specified by RSVP).
Header length: 16 ← Header length of the CAP object.
Element type: 2 ← „downstream receiver“ element.
Element length: 16 ← length of the element.
Sequence number: 0
Element data: 5 ← Total number of downstream receivers.

The list containing charging and accounting information inside the CAP module after the CAP object arrives:

flow handle = 0
tot_recv = 5 ← Total number of downstream receivers.
Upstream...193.175.33.9
Upstream seqno = 0
Upstream vif = 1
Downstream node list ...

Down node...193.175.35.9 ← IP address of the downstream hop passed by RSVP module
Token bucket rate = 10

Token bucket depth = 20
vif = 5
tot_rcv = 5 ← Total number of downstream receivers.

2. Step 2: The CAP module receives a “downstream receivers” CAP element contained in a CAP object from another downstream hop:

Length: 32
Class: 14
Type: 1
Header length: 16
Element type: 2
Element length: 16
Sequence number: 0
Element data: 4

The list containing charging and accounting information inside the CAP module after the CAP object arrives:

flow handle = 0
tot_rcv = 9 ← Total number of downstream receivers.
Upstream...193.175.33.9
Upstream seqno = 0
Upstream vif = 1
Downstream node list ...

Down node...193.175.36.9 ← IP address of the downstream hop passed by RSVP module
Token bucket rate = 10 ← Traffic parameter.
Token bucket depth = 40 ← Traffic parameter.
vif = 6
tot_rcv = 4 ← Total number of downstream receivers.

Down node...193.175.35.9
Token bucket rate = 10 ← Traffic parameter.
Token bucket depth = 20 ← Traffic parameter.
vif = 5
tot_rcv = 5 ← Total number of downstream receivers.

3. Step 3: The CAP module receives “downstream receivers” CAP elements contained in CAP objects from the downstream hops notifying the total number of receivers changed and from a new downstream hop:

„193.175.35.9“:
Length: 32
Class: 14
Type: 1
Header length: 16
Element type: 2
Element length: 16
Sequence number: 0

Element data: 10*„193.175.36.9“:***Length: 32****Class: 14****Type: 1****Header length: 16****Element type: 2****Element length: 16****Sequence number: 0****Element data: 11***„193.175.37.9“:***Length: 32****Class: 14****Type: 1****Header length: 16****Element type: 2****Element length: 16****Sequence number: 0****Element data: 12**

The list containing charging and accounting information inside the CAP module after the CAP object arrives:

*flow handle = 0**tot_rcv = 33 ← New total downstream receiver.**Upstream...193.175.33.9**Upstream seqno = 0**Upstream vif = 1**Downstream node list ...**Down node...193.175.37.9 ← New downstream hop.**Token bucket rate = 10**Token bucket depth = 60**vif = 7**tot_rcv = 12**Down node...193.175.36.9**Token bucket rate = 10**Token bucket depth = 40**vif = 6**tot_rcv = 11 ← New total downstream receiver.**Down node...193.175.35.9**Token bucket rate = 10**Token bucket depth = 20**vif = 5**tot_rcv = 10 ← New total downstream receiver.*

4. Step 4: The CAP module receives an „upstream charge“ CAP element contained in a CAP object from the upstream hop and shares the charge among the downstream hops.

Content of the „upstream charge“ CAP element:

Length: 36

Class: 14

Type: 1

Header length: 16

Element type: 1 ← „upstream charge“ CAP element.

Element length: 20

Sequence number: 0 ← „Go Back n“ sequence number of „upstream charge“ CAP element.

Element data: 120.000000 ← The upstream charge.

The list containing charging and accounting information inside the CAP module after the „upstream charge“ CAP element arrives:

flow handle = 0

tot_rcv = 33

Upstream...193.175.33.9

Upstream seqno = 1 ← New sequence number.

Upstream vif = 1

Downstream node list ...

Down node...193.175.37.9

Token bucket rate = 10

Token bucket depth = 60

vif = 7

tot_rcv = 12

Charge list of a downstream node:

seqno = 0

charge = 75.415020 ← The shared charge according to the number of receivers and reserved QoS level. This downstream hop has the highest reserved QoS level and the highest total number of receivers. Thus, it has the highest shared charge.

Down node...193.175.36.9

Token bucket rate = 10

Token bucket depth = 40

vif = 6

tot_rcv = 11

Charge list of a downstream node

seqno = 0

charge = 32.463768 ← The shared charge according to the number of receivers and reserved QoS level.

Down node...193.175.35.9

Token bucket rate = 10

Token bucket depth = 20

vif = 5

tot_rcv = 10

Charge list of a downstream node

seqno = 0

charge = 12.121212 ← The shared charge according to the number of receivers and reserved QoS level. This downstream hop has the lowest reserved QoS level and the lowest total number of receivers. Thus, it has the lowest shared charge.

5. Step 5: Three more “upstream charge” CAP elements contained in a CAP object from the upstream hop arrive.

Content of the „upstream charge“ CAP objects:

Length: 36

Class: 14

Type: 1

Header length: 16

Element type: 1 ← „upstream charge“ CAP element.

Element length: 20

Sequence number: 1

Element data: 120.000000 ← Upstream charge.

Length: 36

Class: 14

Type: 1

Header length: 16

Element type: 1 ← „upstream charge“ CAP element.

Element length: 20

Sequence number: 2

Element data: 240.000000

Length: 36

Class: 14

Type: 1

Header length: 16

Element type: 1 ← „upstream charge“ CAP element.

Element length: 20

Sequence number: 3

Element data: 360.000000

The list containing the charging and accounting information inside the CAP module after CAP elements have arrived:

flow handle = 0

tot_rcv = 33

Upstream...193.175.33.9

Upstream seqno = 4 ← New „Go Back n“ sequence number of the „upstream charge“ element.

Upstream vif = 1

Downstream node list ...

Down node...193.175.37.9

Token bucket rate = 10
Token bucket depth = 60
vif = 7
Charge list of a downstream node
seqno = 0
tot_rcv = 12
charge = 75.415020
seqno = 1
charge = 75.415020 ← New charge to downstream hop.
seqno = 2
charge = 150.830040 ← New charge to downstream hop.
seqno = 3
charge = 226.245059 ← New charge to downstream hop.

Down node...193.175.36.9
Token bucket rate = 10
Token bucket depth = 40
vif = 6
tot_rcv = 11
Charge list of a downstream node
seqno = 0
charge = 32.463768
seqno = 1
charge = 32.463768 ← New charge to downstream hop.
seqno = 2
charge = 64.927536 ← New charge to downstream hop.
seqno = 3
charge = 97.391304 ← New charge to downstream hop.

Down node...193.175.35.9
Token bucket rate = 10
Token bucket depth = 20
vif = 5
tot_rcv = 10
Charge list of a downstream node
seqno = 0
charge = 12.121212
seqno = 1
charge = 12.121212 ← New charge to downstream hop.
seqno = 2
charge = 24.242424 ← New charge to downstream hop.
seqno = 3
charge = 36.363636 ← New charge to downstream hop.

6. Step 6: The CAP module receives an „upstream charge“ CAP element with a wrong sequence number from the upstream hop

Content of the CAP element:

Length: 36

Class: 14

Type: 1

Header length: 16

Element type: 1 ← „upstream charge“ CAP element.

Element length: 20

Sequence number: 0 ← wrong sequence number.

Element data: 120.000000

The list containing charging and accounting information inside the CAP module after the out-of-order „upstream charge“ CAP element arrived: (no change)

flow handle = 0

tot_rcv = 33

Upstream...193.175.33.9

Upstream seqno = 4

Upstream vif = 1

Downstream node list ...

Down node...193.175.37.9

Token bucket rate = 10

Token bucket depth = 60

vif = 7

tot_rcv = 12

Charge list of a downstream node

seqno = 0

charge = 75.415020

seqno = 1

charge = 75.415020

seqno = 2

charge = 150.830040

seqno = 3

charge = 226.245059

Down node...193.175.36.9

Token bucket rate = 10

Token bucket depth = 40

vif = 6

tot_rcv = 11

Charge list of a downstream node

seqno = 0

charge = 32.463768

seqno = 1

charge = 32.463768

seqno = 2

charge = 64.927536

seqno = 3

charge = 97.391304

Down node...193.175.35.9

Token bucket rate = 10

Token bucket depth = 20
vif = 5
tot_rcv = 10
Charge list of a downstream node
seqno = 0
charge = 12.121212
seqno = 1
charge = 12.121212
seqno = 2
charge = 24.242424
seqno = 3
charge = 36.363636

7. Step 7: The CAP module is prompted by RSVP to create a „downstream receivers“ CAP object with acknowledgment for the „upstream charge“ CAP element to the upstream hop.

Content of the “downstream receivers” CAP element:

Length: 32
Class: 14
Type: 1
Header length: 16
Element type: 2 ← „downstream receivers“ CAP element.
Element length: 16
Sequence number: 4 ← Sequence number of the expected „upstream charge“ CAP element.
Element data: 33 ← Total number of downstream receivers.

8. Step 8: The CAP module is prompted by RSVP to create the “upstream charge” CAP elements that are sent to the downstream hops.

Content of the “upstream charge” CAP elements:

“195.173.35.9”:
Length: 116
Class: 14
Type: 1
Header length: 16
Element type: 1
Element length: 20
Sequence number: 0 ← Sequence number.
Element data: 12.121212 ← Shared charge.
Element type: 1
Element length: 20
Sequence number: 1 ← Sequence number.
Element data: 12.121212 ← Shared charge.
Element type: 1
Element length: 20
Sequence number: 2 ← Sequence number.
Element data: 24.242424 ← Shared charge.
Element type: 1

Element length: 20

Sequence number: 3 ← Sequence number.

Element data: 36.363636 ← Shared charge.

“195.173.36.9”:

Length: 96

Class: 14

Type: 1

Header length: 16

Element type: 1

Element length: 20

Sequence number: 0 ← Sequence number.

Element data: 32.463768 ← Shared charge.

Element type: 1

Element length: 20

Sequence number: 1 ← Sequence number.

Element data: 32.463768 ← Shared charge.

Element type: 1

Element length: 20

Sequence number: 2 ← Sequence number.

Element data: 64.927536 ← Shared charge.

Element type: 1

Element length: 20

Sequence number: 3 ← Sequence number.

Element data: 97.391304 ← Shared charge.

“195.173.37.9”:

Length: 96

Class: 14

Type: 1

Header length: 16

Element type: 1

Element length: 20

Sequence number: 0 ← Sequence number.

Element data: 75.415020 ← Shared charge.

Element type: 1

Element length: 20

Sequence number: 1 ← Sequence number.

Element data: 75.415020 ← Shared charge.

Element type: 1

Element length: 20

Sequence number: 2 ← Sequence number.

Element data: 150.830040 ← Shared charge.

Element type: 1

Element length: 20

Sequence number: 3 ← Sequence number.

Element data: 226.245059 ← Shared charge.

9. Step 9: The CAP module receives “downstream receivers” CAP elements with acknowledgement for the shared charges from its downstream hops “193.175.35.9”, “193.175.36.9”, “193.175.37.9”:

“193.175.35.9”:

Length: 32

Class: 14

Type: 1

Header length: 16

Element type: 2

Element length: 16

Sequence number: 4 ← Expected sequence number.

Element data: 10

“193.175.36.9”:

Length: 32

Class: 14

Type: 1

Header length: 16

Element type: 2

Element length: 16

Sequence number: 3 ← Expected sequence number.

Element data: 11

“193.175.37.9”:

Length: 32

Class: 14

Type: 1

Header length: 16

Element type: 2

Element length: 16

Sequence number: 1 ← Expected sequence number.

Element data: 12

The list containing charging and accounting information inside the CAP module after the “downstream receivers” CAP element and acknowledgments for the shared charges have arrived: (Note that some shared charges to the downstream hops have been removed):

flow handle = 0

tot_rcv = 33

Upstream...193.175.33.9

Upstream seqno = 4

Upstream vif = 1

Downstream node list ...

Down node...193.175.37.9

Token bucket rate = 10

Token bucket depth = 60

vif = 7

tot_rcv = 12

Charge list of a downstream node

seqno = 1 ← The shared charge with sequence number 0 has been acknowledged and removed.

charge = 75.415020

seqno = 2

charge = 150.830040

seqno = 3

charge = 226.245059

Down node...193.175.36.9

Token bucket rate = 10

Token bucket depth = 40

vif = 6

tot_rcv = 11

Charge list of a downstream node

seqno = 3

charge = 97.391304 ← The shared charges with sequence number 0, 1, and 2 have been acknowledged and removed.

Down node...193.175.35.9

Token bucket rate = 10

Token bucket depth = 20

vif = 5

tot_rcv = 10 ← The shared charges with sequence number 0, 1, 2, and 3 have been acknowledged and removed.

10. Step 10: CAP module receives a CAP object containing three “upstream charge” CAP elements from the upstream hop

Content of the CAP object:

Length: 76

Class: 14

Type: 1

Header length: 16

Element type: 1

Element length: 20

Sequence number: 3 ← This sequence number is not expected, it is a duplicate.

Element data: 120.000000

Element type: 1

Element length: 20

Sequence number: 4 ← This sequence number is expected.

Element data: 240.000000

Element type: 1

Element length: 20

Sequence number: 5

Element data: 360.000000

After CAP object has arrived:

```

flow handle = 0
tot_rcv = 33
Upstream...193.175.33.9
Upstream seqno = 6
Upstream vif = 1
Downstream node list ...

Down node...193.175.37.9
Token bucket rate = 10
Token bucket depth = 60
vif = 7
tot_rcv = 12
Charge list of a downstream node
seqno = 1
charge = 75.415020
seqno = 2
charge = 150.830040
seqno = 3
charge = 226.245059
seqno = 4
charge = 377.075099 ←New shared charge to the downstream node.

```

```

Down node...193.175.36.9
Token bucket rate = 10
Token bucket depth = 40
vif = 6
tot_rcv = 11
Charge list of a downstream node
seqno = 3
charge = 97.391304
seqno = 4
charge = 162.318841 ←New shared charge to the downstream node.

```

```

Down node...193.175.35.9
Token bucket rate = 10
Token bucket depth = 20
vif = 5
tot_rcv = 10
Charge list of a downstream node
seqno = 4
charge = 60.606061 ←New shared charge to the downstream node.

```

C.2 Testing of the Implementation in the real RSVP environment

The test scenario in the real RSVP environment is organized as follows:

In the network of GMD Fokus, a host named rockmaster is used as a sender and three hosts named dagobert, balu, and tao are used as receivers of a multicast session. The receivers join the multicast session one after another with different QoS levels. This scenario results in different charges to the receivers depending on the current number of receivers in the

multicast session and the QoS they receive. The IP addresses of rockmaster, dagobert, balu, and tao are 193.175.133.130, 193.175.132.175, 193.175.133.74, and 193.175.133.51 respectively.

A small piece of a log file at the sender rockmaster is shown here to demonstrate the functionality of our charging and accounting protocol in the real environment of RSVP.

-----flow handle = 0

Upstream... is API.

Upstream seqno = 0

Upstream vif = 1

Total number of receivers = 1

Downstream node list ...

Downstream node: 193.175.132.175

QoS:

Token bucket rate (B/s): 10.000000

Token bucket depth (B) : 10.000000

Peak data rate (B/s): 10.000000

Min Policed Unit (B): 10

Max pkt size (B): 65535

vif = 0

tot_rcv = 1

Charge list of downstream node

seqno = 0

charge = 100.000000

-----flow handle = 0

Upstream... is API.

Upstream seqno = 0

Upstream vif = 1

Total number of receivers = 2

Downstream node list ...

Downstream node: 193.175.133.74

QoS:

Token bucket rate (B/s): 9.000000

Token bucket depth (B) : 9.000000

Peak data rate (B/s): 9.000000

Min Policed Unit (B): 9

Max pkt size (B): 65535

vif = 0

tot_rcv = 1

Charge list of downstream node

seqno = 0

charge = 40.500000

Downstream node: 193.175.132.175

QoS:

Token bucket rate (B/s): 10.000000

Token bucket depth (B) : 10.000000
 Peak data rate (B/s): 10.000000
 Min Policed Unit (B): 10
 Max pkt size (B): 65535
 vif = 0
 tot_rcv = 1
 Charge list of downstream node
 seqno = 1
 charge = 59.500000

-----flow handle = 0

Upstream... is API.
 Upstream seqno = 0
 Upstream vif = 1
 Total number of receivers = 3
 Downstream node list ...

Downstream node: 193.175.133.51
 QoS:
 Token bucket rate (B/s): 9.000000
 Token bucket depth (B) : 9.000000
 Peak data rate (B/s): 9.000000
 Min Policed Unit (B): 9
 Max pkt size (B): 65535
 vif = 0
 tot_rcv = 1
 Charge list of downstream node
 seqno = 0
 charge = 27.000000
 seqno = 1
 charge = 27.000000

Downstream node: 193.175.133.74
 QoS:
 Token bucket rate (B/s): 9.000000
 Token bucket depth (B) : 9.000000
 Peak data rate (B/s): 9.000000
 Min Policed Unit (B): 9
 Max pkt size (B): 65535
 vif = 0
 tot_rcv = 1
 Charge list of downstream node
 seqno = 1
 charge = 27.000000

Downstream node: 193.175.132.175
 QoS:
 Token bucket rate (B/s): 10.000000
 Token bucket depth (B) : 10.000000
 Peak data rate (B/s): 10.000000

Min Policed Unit (B): 10
Max pkt size (B): 65535
vif = 0
tot_recv = 1
Charge list of downstream node
seqno = 2
charge = 46.000000

Appendix D References

- [Armi96] G. Armitage, Support for Multicast over UNI 3.0/3.1 based ATM Networks. Internet Request for Comments, RFC 2022, November 1996.
- [ATM94] Interim Inter-switch Signaling Protocol (IISP) Specification v1.0. The ATM Forum, December 94.
- [ATM96] Private Network-Network Interface Specification Version 1.0 (PNNI 1.0). The ATM Forum, March 1996.
- [Bake96] F. Baker, RSVP Cryptographic Authentication, Internet Draft, draft-ietf-rsvp-md5-02.txt, 1996.
- [Brad97] R. Braden, L. Zhang, S. Berson, S. Herzog and S. Jamin. Resource ReSerVation Protocol (RSVP). Internet Request for Comments, RFC 2205, September 1997.
- [Come95] D. Comer. Internetworking with TCP/IP Volume I (third edition), Prentice Hall, 1995.
- [CoSS97] A. Corghi, L. Salgarelli, H. Sanneck, M. Smirnov, D. Witaszek. Design Experiences with IP Multicast Integrated Services over ATM, 1997.
- [CoSS98] A. Corghi, L. Salgarelli, H. Sanneck, M. Smirnov, D. Witaszek, T. Zseby. Specification of the IP/ATM Integration in the Multicube Context, February 1998.
- [CaCS98] M. Canosa, A. Corghi, H. Sanneck, M. Smirnov, L. Vismara, D. Witaszek, T. Zseby. Development of early-stage prototype IP/ATM integration, April 1998.
- [CaSZ98] G. Carle, M. Smirnov, T. Zseby. Charging and Accounting Architecture for IP Multicast Integrated Services over ATM. Fourth International Symposium on Interworking, Ottawa, Canada, July 6-10, 1998.
- [Deer89] S. Deering. Hosts Extensions for IP Multicast. Internet Request for Comments, RFC 1112, August 1989.
- [Deer97] S. Deering, D. Estrin, D. Farinacci, V. Jacobson, C. Liu, and L. Wei. Protocol Independent Multicast Version 2, Dense Mode Specification, Internet Draft, draft-ietf-idmr-pim-dm-spec-05.ps, May 1997.
- [Est97] D. Estrin, D. Farinacci, A. Helmy, D. Thaler, S. Deering, M. Handley, V. Jacobson, C. Liu, P. Sharma, and L. Wei. Protocol Independent Multicast-Sparse Mode (PIM-SM): Protocol Specification. Internet Request for Comments, RFC 2117, July 1997.
- [Fird97] E. Firdman. Rx for the Internet: Usage-based Pricing. Datacommunications, January 1997. Available at http://www.data.com/business_case/rx_internet.html.
- [Hals95] F. Halsall. Data Communications, Computer Networks and Open Systems (fourth edition), Addison-Wesley Publishing Company, 1995.

- [Herz96a] S. Herzog. Accounting and Access Control for Multicast Distributions: Models and Mechanisms, Ph.D. thesis, University of Southern California, August 1996.
- [Herz96b] S. Herzog. Local Policy Modules (LPM): Policy Control for RSVP. Internet Draft, draft-ietf-rsvp-policy-lpm-01.[ps,txt], November 1996.
- [Luci97] J. V. Luciani, G. Armitage, J. Halpern. Server Cache Synchronization Protocol (SCSP). Internet Draft, draft-ietf-ion-sscp-00.txt, Expires July 1997.
- [Mauf97] T. Maufer, C. Semeria. Introduction to IP Multicast Routing. Internet Draft, draft-ietf-mboned-intro-multicast-03.txt, July 1997.
- [Moy94] J. Moy. Multicast Extensions to OSPF. Internet Request for Comments, RFC 1584, March 1994.
- [Rap98a] S. Herzog. RSVP Extensions for Policy Control. Internet Draft, draft-ietf-rap-rsvp-ext-00.txt, April 1998.
- [Shen96] S. Shenker, D. Clark, D. Estrin, and S. Herzog. Pricing in Computer Networks: Reshaping the Research Agenda. Telecommunications Policy, 20(1), 1996. Also published in Proceedings of the Twenty Third Annual Telecommunications Policy Research Conference, 1995.
- [Smir97] M. Smirnov. EARTH - EAsy IP Multicast Routing THrough ATM clouds, Work-in-progress, Internet Draft, draft-smirnov-ion-earth-02.txt, March 1997.
- [Song97] David Songhurst. Charging Mechanism and Policy: Adapting to the Commercial Environment. IEE Colloquium on Charging for ATM, London, U.K., November 1997.
- [Spra91] J. D. Spragins. Telecommunications: Protocols and Designs, Addison-Wesley Publishing Company, 1991.
- [Stev93] W. R. Stevens. TCP/IP Illustrated Volume I: The Protocols, Addison-Wesley Publishing Company, 1993.
- [Tane96] A. S. Tanenbaum. Computer Networks (third edition), Prentice Hall, 1996.
- [Thom96] S. A. Thomas. IPng and the TCP/IP Protocols, Wiley Computer Publishing, 1996.
- [Wait88] D. Waitzman, C. Partridge, and S. Deering. Distance Vector Multicast Routing Protocol. Internet Request for Comments, RFC 1075, October 1988.
- [Wroc97] J. Wroclawski. The Use of RSVP with IETF Integrated Services. Internet Request for Comments, RFC 2210, September 1997.