Tunnel Agents for Enhanced QoS in the Internet

Hermann de Meer[1,3], Antonio Puliafito[2,4],

Jan-Peter Richter[1,3], and Orazio Tomarchio[2,4]

[1] Dept. of Computer Science, University of Hamburg
Vogt-Kölln-Str. 30, 22527 Hamburg - Germany
{demeer,richter}@informatik.uni-hamburg.de

[2] Istituto di Informatica, Università di Catania
Viale A. Doria 6, 95025 Catania - Italy
{ap,tomarchio}@iit.unict.it

Abstract

Originally, the Internet delivered best-effort service quality with respect to end-to-end delay. Recently, extensions such as RSVP have been proposed to provide guaranteed real-time services as well. Unfortunately, network resources, such as routers, do not yet fully support RSVP reservation protocols so that guarantees cannot truly be given. In this paper, we suggest to follow the paradigm of open programmable networks for a more complete QoS provisioning. Reservation gaps or tunnels are dynamically closed by means of a software-agent approach that is flexibly deployed for an application oriented QoS support. Agents are dynamically located to such tunnels in order to monitor the tunnels, to provide feed-back information in case of QoS violations, and to decide on possible compensating measures to be taken. One such possible measure is redistribution of user specified QoS budgets, like end-to-end delay, among the involved RSVP-capable network components. Re-routing support is another feature achievable by exploiting mobility capabilities of the agent approach based on our Java platform.

Keywords: QoS management, RSVP, Internet, mobile agents, open programmable networks.

---

[3] Supported by a grant from *Deutsche Akademische Austauschdienst (DAAD)*

[4] Supported by a grant from *Conferenza Permanente dei Rettori delle Unversità Italiane (CRUI)*

1 Introduction

Quality of service is an increasingly important issue for distributed multimedia systems. Generally, multimedia applications are very resource demanding so that efficient usage of resources is mandatory, often resulting in extensive resource sharing. On the contrary, hard or soft quality-of-service guarantees, such as timely delivery of data, have to be given. Guarantees are usually implemented by some sort of resource reservation schemes, both in time and space. While efficiency and guarantee are equally important properties, they are driven by contradicting forces so that for a real system choices have to be made to what extent one or the other characteristic should be supported. Constraints are implied by the available resources and the need to customize services according to the requirements imposed by specific applications and users' preferences.

The Internet was originally designed to provide a guaranteed reliable communication service, assuming some fault model. Routing algorithms are applied that are adaptive to node failures and a transport protocol is used that handles occasional losses and corruptions of data packets. Other qualities and guarantees were considered to be of less or of negligible importance. Time, in particular, was assumed to be abundantly available so that a best-effort service quality resulted with respect to end-to-end delay. Recently, extensions to the Internet such as the resource reservation protocol (RSVP) [18, 2] have been proposed to provide guaranteed real-time services as well while simultaneously relaxing the requirements with respect to reliable communication. Unfortunately, network resources, such as routers, do not yet fully support RSVP reservation protocols so that guarantees cannot truly be given.

RSVP is based on the concept of a software layer running on top of IP, thereby using underlying routing protocols and providing admission and policing mechanisms on a per application basis. RSVP is designed to operate inside non-fully RSVP-based networks. If non-RSVP-capable nodes belong to the path of an established session for which the user wants a certain QoS, RSVP tries to reserve adequate resources on the nodes where this is possible and relies on a best-effort strategy in the remaining cases. Of course, in such cases no global guarantees can be given.

In this paper, we propose an approach dealing with *tunnels* leading through a so-called *cloud* of non-RSVP-capable network entities and how to manage them with respect to application specific QoS requirements. Note that in our terminology a tunnel refers to those routers which are assigned to an application specific path through a cloud. The goal is to provide further enhanced QoS than currently possible with present Internet technology including reservation mechanisms [18].

We suggest following the paradigm of open programmable networks for a more complete QoS provisioning [12]. Reservation gaps or tunnels are dynamically closed by means of a software-agent approach that is flexibly deployed for an application oriented QoS support. Agents are dynamically located to such tunnels in order to monitor the tunnels, to provide feed-back information in case of QoS violations, and to decide on possible compensating measures to be taken. One such possible measure is redistribution of user specified QoS budgets, like end-to-end delay, among the involved RSVP-capable network components. Re-routing support is another feature achievable by exploiting mobility capabilities of the agent approach based on our Java platform.

Such mechanisms are implemented in a completely distributed fashion and are very well suited to the use of agent technology. The concept of *agent* (mobile, autonomous or intelligent) has recently gained much attention in several research areas, from artificial intelligence to distributed systems, from computer networks to software engineering [3, 7]. Software agents are already arranged for operating in a distributed manner and they permit very flexible communication and co-operation schemes, suggesting themselves as an attractive foundation for open programmable networks.

The remainder of the paper is organized as follows. Section 2 provides the technical prerequisites and introduces the tunnel problem in more detail. Section 3 shortly reviews our agent-based QoS reference architecture. Details of our solution approach based on tunnel agents are given in section 4 and a closer look at implementation issues is taken in section 5. Finally, the paper is concluded in Section 6.

2 Limitations of QoS provisioning in IP networks

Heterogeneous networking will become a reality in the near future. Even though new technologies like ATM inherently provide some means of guaranteed QoS provisioning, protocols from the TCP/IP world will continue to be used. This means that QoS provisioning has also to be performed in the Internet world. A currently popular approach is based on RSVP [2, 18]. Unfortunately, there are certain limitations coming with the introduction of RSVP. First of all, it is unlikely that RSVP will be supported by all routers in the foreseeable future [2]. Second, RSVP provides rather basic functionalities that leave room for more application oriented support for QoS provisioning. Additional effort will be necessary here to provide a more complete solution. We propose to use agent technologies based on the concept of open programmable networks to this end. Although being particularly taylored towards complementing RSVP as a defective IP-based reservation mechanism, the agent approach seems to be a promising compensation measure well complementing *other* incomplete reservation technique that may come up in the future.

2.1 RSVP

RSVP is a framework for resource reservation and QoS provisioning mechanisms within the Internet. The framework is an extension of classical IP routers, including the processing of RSVP messages as well as a more controlled operation of packet forwarding. While the RSVP protocol merely defines the exchange of control messages to build up and maintain a shared knowledge of reservation state, several QoS supporting *integrated services* are defined within that framework that complement the protocol with detailed definitions of router behaviour. Two such services have been defined: controlled-load service [17] and guaranteed service [15]. The former service is characterized by the integrated services working group [17] as follows:

> Controlled-load service provides the client data flow with a quality of service closely approximating the QoS that same flow would receive from an unloaded network element, but uses capacity (admission) control to assure that this service is received even when the network element is overloaded.

While this form of QoS provision may be sufficient for self-adapting applications, more advanced reservations schemes are necessary if QoS guarantees are actually to be given.

The latter service is characterized [15] as follows:

> Guaranteed service provides firm (mathematically provable) bounds on end-to-end datagram queueing delays. The service makes it possible to provide a service that guarantees both delay and bandwidth.

We apply our agent-based approach to solve a specific problem in the deployment of guaranteed service within the framework of RSVP. More details will be given in the following section.

2.2 The tunnel problem

Since the introduction of RSVP requires major changes to the operation of IP routers, not all routers within the Internet will be capable of RSVP support in the near or mid future. In fact, the authors of [2] admit that RSVP support may never be ubiquitous, i.e., non-RSVP-capable routers will remain a reality. For the sake of brevity, we will refer to RSVP-capable routers and non-RSVP-capable routers as *RSVP routers* and *non-RSVP routers*, respectively, in the sequel.

The RSVP protocol that handles the exchange of control messages simply ignores non-RSVP routers. RSVP messages are forwarded through a *cloud* of non-RSVP routers and reservation is merely performed outside the cloud. However, since non-RSVP routers may degrade the perceived QoS in an uncontrolled way, no end-to-end guarantee can be given. It is still believed that the application of RSVP on the remaining part of the path is beneficial, because the performance of the non-RSVP routers *may* be sufficient to fulfil the demanded level of QoS.

Besides the ability to communicate control messages of the RSVP protocol through such a cloud of non-RSVP routers, no further action is taken within the RSVP framework to handle such a situation. Instead, the behaviour of the routers within the tunnel is simply ignored and the application is given an indication that the QoS values are to be understood as approximate.

For the sake of a clear terminology, we define the set of routers that comprise the sub-path within a cloud of non-RSVP routers to be a *tunnel*; a single router in the tunnel is referred to as a *tunnel router*. It is our goal to improve the end-to-end quality by monitoring the tunnel and providing feed-back for an enhancement of the reservation scheme. The overall procedure we discuss in the paper is based on the existence of distributed mechanisms that are able to implement cooperative monitoring techniques in order:

- to monitor the behavior of the system in terms of the performance level it provides;
- to use information on the status of the system for establishing a QoS negotiation phase and for proper distribution of application requirements on system resources;
- to interact with the reservation entities within the RSVP routers.

The basic idea of the approach consists in exploiting the interaction among properly defined software entities continuously running inside the system in order to implement QoS monitoring, reservation and adaptation functionalities [10].

2.3 An agent-based approach

The crucial part of our reference scenario is the tunnel which represents an entity with a non-predictable QoS behavior. However, if adequate monitoring of the tunnel behavior is carried out, information can be spread out in the system and network adaptation is carried out to compensate for excessive QoS budget consumption within the tunnel.

To pursue this goal we believe that agent technology is a good candidate due to the high level of flexibility it offers, the inherent capacity of remote process execution and the possible high level of interaction. An agent [7] can be defined as a software module, eventually equipped with artificial intelligence mechanisms, which is capable, autonomously or in cooperation with other agents, of solving a certain problem or carrying out a certain task. The more an agent is autonomous, user-friendly, capable of dealing with unexpected situations, capable of learning and improving its performance on the basis of its mistakes, observations, or instructions from the outside, the more intelligent it is considered to be. An agent, therefore, has to be:

- *autonomous*, i.e., it is capable of operating without a direct human action and it performs a certain level of control of its own actions;
- *competent*, i.e., it is capable of properly managing a given situation and improving its behaviour by accumulating experience in specific situations;
- *reliable*, i.e., capable of guaranteeing that its evolution will lead to choices and behaviour that will conform to users' preferences.

Some researchers – especially those who work in the field of artificial intelligence – claim that there are other desirable characteristics, which lead to a *stronger notion* for the concept of agent. *Personality* and *adaptivity* are some of such characteristics but *mobility* is probably one of the most important ones [5, 7]. The use of mobile agents, i.e., agents capable of migration from one machine to another within a heterogeneous computer network, can contribute to building distributed applications more simply and effectively at the same time. Some areas where such technology is widely applied are:

- *Distributed Management:* mobile agents enable the system designer to delegate some management functions from a central station to remote nodes, thus reducing the workload on the central station and improving the exploitation of the available resources [8];
- *Distributed Computation:* thanks to the possibility to pilot the node on which mobile agents are executed, agent technology offers a new paradigm for parallel computation in a distributed network of workstations;
- *Collaborative Applications:* mobile agents provide an effective support for sharing data and any kind of documents; they offer a flexible architecture and enable users to share various network resources.

In [13, 4] we proposed a reference architecture for QoS management with software agents. Implementation issues have also been discussed in [14], where a Java-based approach was proposed to deal with agent migration inside a heterogeneous distributed environment. The flexibility and the generality of this agent-based architecture for QoS management is exploited to practically solve the tunnel problem. Autonomy, distributed computation and mobility are the main characteristics of our agents. A short review of our QoS reference architecture is provided in Section 3.

3 Reference Architecture

The use of an agent-based infrastructure for the management of QoS is a good choice for implementing several adaptation strategies by which available resources need to be re-arranged in such a way that the user is still satisfied. Particularly interesting for the implementation of adaptation strategies is the exploitation of agents' cooperation abilities for optimization purposes [6]. Although reasoning and decision are locally performed by each agent, the agents are coordinated by communicating certain events and internal status information to each other. As a result, each agent is influenced by and can influence the behavior of other agents.

According to the specific problem, the basic functionalities that a software agent may require can be summarized as follows:

- to monitor system status by combining and filtering data to reduce message overhead between managed components and management applications;
- to cooperate with other agents by exchanging data and synchronization messages;
- to migrate from one node to another and execute remotely with a transparent use of the available hardware/software resources.

To implement agent retrievability and mobility, a computing platform has been developed [14] which, in order to provide a sort of homogeneity among different hardware and software platforms, adopts Java as its underlying framework. Java [9] is an object-oriented language which allows a modular development of the system: users can write their own agents deriving them from the classes provided by the platform with limited effort. It guarantees portability of code across an always increasing number of software and hardware platforms.

A specialized version of such platform has been developed for the purpose of QoS management [13, 4], which comprises specific nodes to collect QoS applications. We identify such nodes as *Servers of Applications* and assume that they store QoS applications to be eventually downloaded and remotely executed on demand.

The reference architecture we refer to is depicted in Fig. 1. Software agents are classified in the architecture according to the specific functionalities they carry out. A user on the client system interacts with the QoS management component by means of an appropriate *User Agent*. This agent is specific for each service required by the user, and enables the user to specify and negotiate the desired QoS parameters. The *System Agent* provides all the information about the client's system state; relying on the services provided by the operating system of the host machine, it has the mechanisms to control the QoS on the client. On each server node a *Service Agent* is present which has to monitor whether the QoS parameters (for each service required

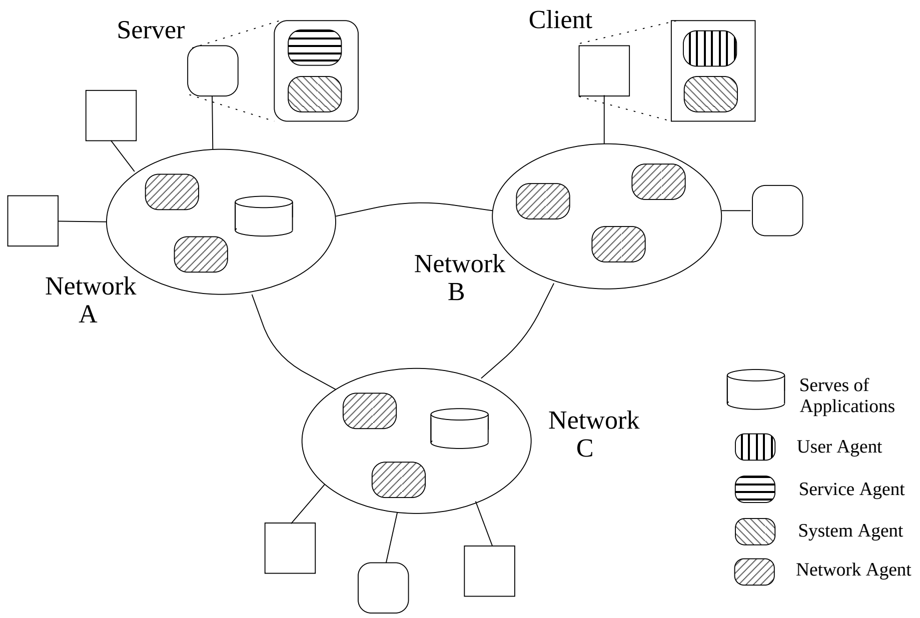

Figure 1: Reference Architecture

by the clients) are respected. It keeps track of the users logged in and their occupied resources, in order to optimize the use of available resources. This agent also maintains the knowledge of the server's capabilities.

Control functionalities are implemented by *Network Agents*. Network agents have the following responsibilities:

- reservation of suitable network resources;
- monitoring of system parameters;
- initiation of corrective QoS adaptation activities if QoS violations are detected;
- interaction with the network management system in order to obtain additional resources and/or to optimize their use.

In the next section the reference architecture is specifically applied to the solution of the tunnel problem.

4 Tunnel Agents

In this section we focus our attention on the behavior of *tunnel agents*. Their primary goal is the monitoring of tunnel properties in order to interact with the RSVP routers for assuring the end-to-end QoS to the user.

4.1 Tunnel setup

When an application on a receiving node makes a request for a reservation, RSVP carries the request through the network by visiting each node that belongs to the routing path between the receiving machine and the sender. On each intermediate node RSVP tries to reserve adequate resources to guarantee the desired QoS specified by the user. If there are enough resources on each router and if the admission test succeeds, the data stream starts flowing on the pre-established path with the negotiated QoS. However, some non-RSVP nodes, referred to as a tunnel, may be located along the route (Fig. 2).

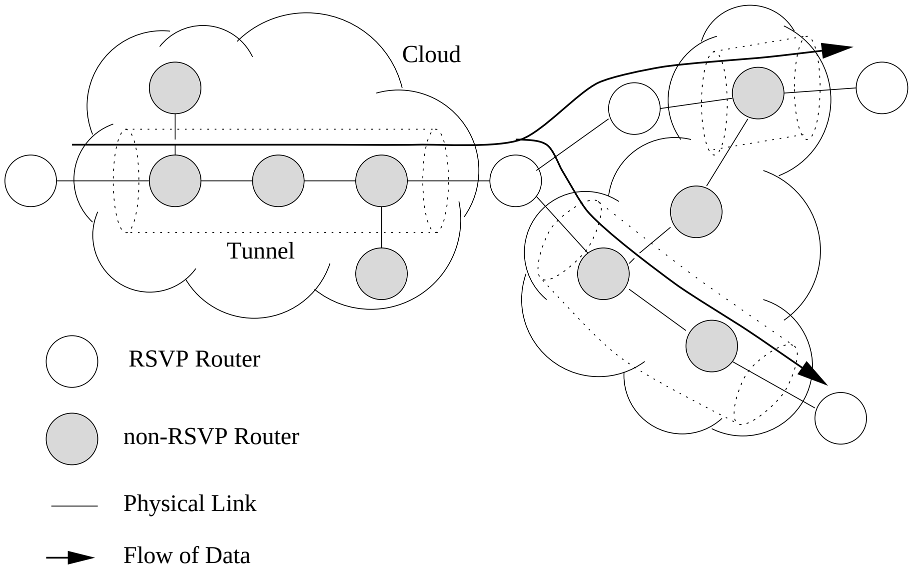

Figure 2: Tunnels

In order to monitor the tunnel behavior and obtain useful performance indices for characterization of the delay experienced in crossing the tunnel, we adopt the agent approach described in the previous section. Two particular network agents, in the following referred to as the *tunnel agents*, are associated to each tunnel. More specifically, tunnel agents are located on the RSVP nodes at the edges of each tunnel, as shown in Fig.3. The RSVP nodes at the edges of a tunnel will be referred to as *edge routers* in the sequel. Note that an RSVP router may be an edge router for two adjacent tunnels if it is surrounded by non-RSVP routers. In this case two instances of a tunnel agent are located on that router. We assume that each RSVP node is able to execute our agents. Furthermore, all nodes, including non-RSVP nodes, are assumed to support standard network management protocols like SNMP [16] to provide some means for monitoring. In an extreme scenario, none of the routers supports RSVP and thus the tunnel spans over the entire path. In this case, the tunnel agents are located on the application hosts and provide the application with useful performance monitoring data to allow for adaptation of traffic volume and encoding. The presence of tunnel agents running on edge routers is not punishing, although the edge routers are already performing RSVP functionalities. The events managed by tunnel agents are order of magnitude less frequent than the continuous packets forwarding activity performed by a router.

The code of the agents is maintained in a generic node which plays the role of a server of QoS applications. Once a tunnel is identified, the user agent located on the receiving application node sends a request to the nearest QoS application server to upload the tunnel agent code, which is then dynamically executed (for further details on these mechanisms see [14]). The main advantage of such an approach consists in avoiding installation of the code on each node: the location of a tunnel depends on dynamic factors like the existence and location of the session path as well as more static properties like the RSVP capability of the nodes. It may even happen that a node will never be at the edge of a tunnel. A tunnel agent installed on such a node would not be useful and would only consume node resources. The existence of simple mechanisms for code retrieval and downloading is thus a useful feature for our specific problem. Agent migration is also exploited when, due to some reason, the dynamic routing algorithms of the Internet cause a relocation of the session path. Migration of an already instantiated tunnel agent carries some knowledge of flow requirements to the new node. Operations performed by tunnel agents will be described in the next two sections.

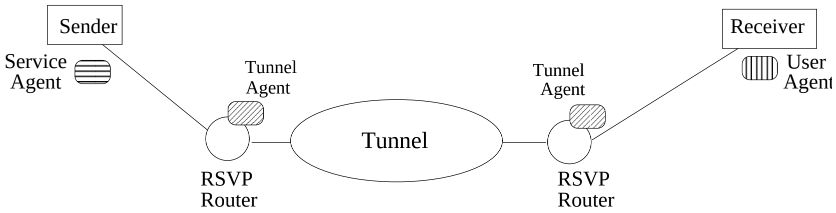

Figure 3: Tunnel Agents

The set-up procedure proceedes according to the following steps:

1. The receiving application becomes aware of the existence of non-RSVP routers along the path. This is part of the standard RSVP functionality. Subsequently, the application triggers its user agent to initiate the tunnel setup procedure.

2. As a second step, the exact location of the tunnel or tunnels must be detected. There are at least three different ways to achieve this goal.

   (a) By using SNMP, the user agent queries the RSVP routers for their respective RSVP neighbours and their direct IP connectivity. If an RSVP router has no direct IP connectivity to one of its RSVP neighbours, it must be an edge router. Following the chain of RSVP neighbours, the user agent eventually learns about the complete sets of RSVP routers and edge routers along the path. Since an SNMP request/reply involves only two UDP packets, it will be the best solution for small networks. However, this method has the disadvantage that it scales quite badly, so it may only be appropriate for short end-to-end paths.

(b) A *discovery agent* is sent out from the host of the receiving application. This agent migrates from RSVP router to RSVP router and issues local SNMP requests to learn about the RSVP neighbourhood relation and the directly connected routers. Finally the agent returns the collected information back to the application host. This solution needs only linear time and is therefore more suitable for longer end-to-end paths.

(c) A third possible implementation of the tunnel detection step could be an extension of the RSVP code that is installed on the RSVP routers. Since each RSVP router is aware of being an edge router or not, each edge router could send a special message to the application host. However this requires the extension of the functionality provided by the RSVP module on each router, which may not be easily achievable.

A selection of a tunnel discovery method between the first and the second method can easily be done adaptively based on the number of RSVP hops.

3. An instance of a tunnel agent is uploaded to each edge router.

4. Each tunnel agent that is located on an upstream edge router starts a traceroute procedure to learn about the non-RSVP routers constituting its tunnel. This information is communicated to the corresponding downstream tunnel agents[1].

5. All tunnel agents monitor the tunnel behaviour and eventually interact with the RSVP protocol, as described in the following sections.

Steps 1 and 2 need more explanation.

RSVP packets are sent as raw IP datagrams with the destination IP address of the path end-point. Furthermore, each RSVP Path message and each RSVP Resv message also carry the IP sender address of the last RSVP router. RSVP packets are slightly modified each time they cross an RSVP router. Especially, the IP-TTL is copied into the Send_TTL field of the RSVP header. A non-RSVP router does not modify an RSVP message but just forwards it to

[1] A multicast tunnel may have more than one downstream edge.

the following router along the path with a decremented IP-TTL. If a receiving RSVP router discovers a mismatch between IP-TTL and Send_TTL then this is taken as an indication for the existence of at least one non-RSVP router between the last RSVP router and itself. The RSVP router is then aware of the existence of a tunnel and also of the IP address of the edge router at the other side of the tunnel. This information is subsequently retrievable by SNMP [1]. The implementation of Steps 3 to 5 is straightforward.

4.2 Interaction of tunnel agents with tunnels

An essential task of tunnel agents is to make decisions on how network reconfiguration can be achieved. These decisions can be made only based on information concerning the structure and state of the tunnel. The topological structure of the tunnel, i.e., the set and connectivity of non-RSVP routers, is determined using a traceroute procedure and SNMP, as described in the last section. SNMP is also used to monitor basic performance measures of the tunnel routers like interface utilization, discard rates, or queue lengths by polling the respective MIB entries.

Additionally, the traffic that goes through the tunnel and the QoS it receives is constantly monitored by the set of agents at the edges of the tunnel using cooperative monitoring techniques. As an example, a pair of tunnel agents is able to measure tunnel-end-to-tunnel-end delay by sending time-stamped probe messages. Note that messages, which are time-stamped at the sender side of the entire path, cannot be used to measure tunnel-end-to-tunnel-end delay without additional message exchange between the tunnel agents.

Note that inter-agent communication is not as sensitive to QoS degradations as the realtime user data. Since control traffic is mostly sensitive to loss it can be protected by the methods of timeout and retransmission well-established in the Internet. A slower transmission of a control packet or the additional delay caused by its retransmission in case of a loss will cause the adaptation mechanism to work *slower*. However, after a short time the adaptation will be carried out, limiting the QoS degradation to be a *transient* one. The transmission of control traffic can also be enhanced by using forward error correction techniques. By using a redundant encoding scheme, a limited number of losses can be tolerated. Since the bandwidth demand of the control traffic is much lower than that of user data, this additional traffic is not very likely to enhance network congestion further. Finally, note that much of the control traffic will be exchanged between the (last) downstream tunnel agent and the user agent on the receiving node as well as between tunnel agents of neighbouring tunnels. Since this traffic runs entirely over RSVP-capable nodes, it can be highly prioritized and thus protected from QoS degradations.

4.3 Interaction of tunnel agents with RSVP

The total end-to-end delay is the sum of all delays encountered along the transmission path. Therefore, a bound on end-to-end delays required by an application is considered as a QoS budget that can be arbitrarily spent by the involved intermediate nodes.

Since the RSVP guaranteed service is based on a fluid flow and leaky-bucket approach, the delay budget is distributed using a reservation of a minimum service rate R. Each RSVP router reserves resources sufficient to support rate R. By this, each local queueing delay is bounded in a way that the sum of all local queueing delays is bounded to a given value. For a detailed discussion see [15]. The rate R is calculated by the receiver in a way that the resulting end-to-end delay meets the QoS demands. However, without agent support this calculation would be done based on the assumption that all routers along the path were RSVP-capable even if the presence of non-RSVP routers was signaled.

To the contrary, in our agent-based approach a higher rate $R' = R + \delta$ is used to set aside some additional delay budget that will not be distributed among the RSVP routers. The calculation of R' is done by the user agent in cooperation with the application and the state information provided by the tunnel agents. The additional delay budget which is characterized by the difference between the original rate R and the announced rate R' is subsequently also communicated to the tunnel agents. If more than one tunnel is present along the path between sender and receiver all tunnel agents agree on a distribution of the additional delay budget using a distributed inter-agent negotiation procedure.

By monitoring the tunnel behaviour, the agents verify if the QoS budgets currently consumed by the tunnel matches the value that was assigned to that tunnel. In case of a mismatch, the tunnel agent initiates horizontal re-negotiation. First, the tunnel agent re-negotiates the partitioning of the set-aside delay budget by re-iteration of the inter-agent negotiation procedure. If there is only one tunnel or the inter-agent re-negotiation was not successful, the QoS parameters in the RSVP routers along the path are re-negotiated. This is done by communicating the additional need of delay budget to the user agent that re-calculates R' and informs the application to change its RSVP reservation messages accordingly.

By re-assignment of budgets compensation for the adverse effect of the tunnel routers is attempted. Of course, the opposite is also true: if the performance of the tunnel enhances as a result of traffic fluctuation, the horizontal re-negotiation is initiated to relax unnecessary stringent requirements on the RSVP routers.

The phase of re-negotiation may fail due to lack of resources. In this case, an indication is sent to the application by the standard RSVP protocol mechanisms. In turn, the application may adapt to the lower level of QoS or may simply terminate.

4.4 Tunnel agent migration

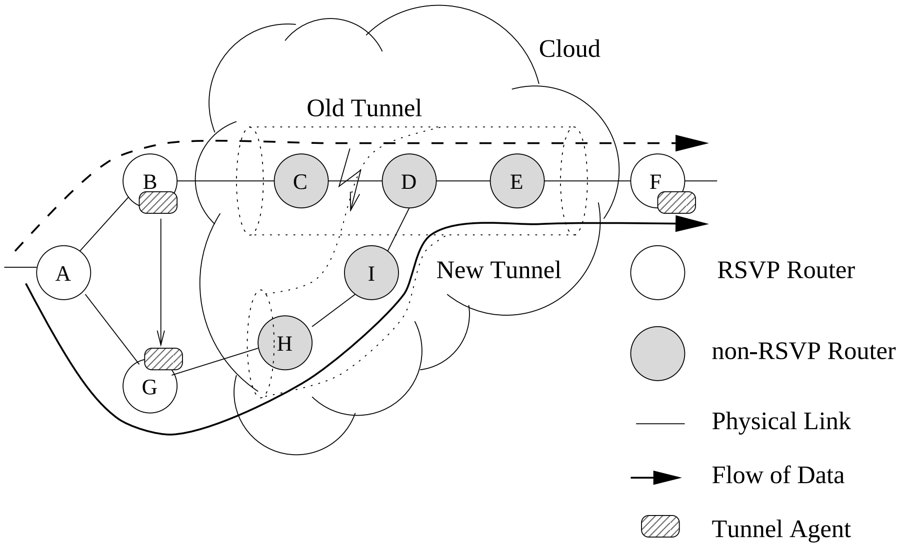

Figure 4: Agent migration after route change

Routing in the Internet is a dynamic process and a route may change during connection time. In this situation the ability of agents to migrate between routers can be exploited. Consider a situation as depicted in Fig.4. A stream of data flows from router A through B, C, D, E to F. When the connection between C and D breaks down, the usual routing algorithms of IP-based networks will find an alternative route, say via G, H and I. Also the mechanisms of the RSVP protocol that strongly rely on IP routing follow this route change and again try to establish reservation state along the new path.

The tunnel agent located on B will notice that it is no longer at the edge of an active tunnel and initiates a relocation procedure. This procedure is carried out in cooperation with the agent on the other end of the tunnel or with the user agent on the application host, depending on whether the upstream or the downstream end of a tunnel has moved. By applying algorithms similar to the tunnel detection procedure, the new topology of the tunnel is discovered. In the case that only one edge of the tunnel has moved, the tunnel agent initiates its own migration to the new location of the tunnel edge. If RSVP routers are located along the new path between G and F, the new tunnel actually consists of a sequence of tunnels. In this case new tunnel agents must be started in addition to migration of one tunnel agent.

In the example, tunnel routers D and E both belong to the old route as well as to the new route and therefore comprise part of the new tunnel. By migrating the tunnel agent from router B to router G, much information about the old tunnel can be saved and re-used to manage the new tunnel. Especially topological information on the unmodified parts and to a certain extent also load information on those parts can be kept.

Of course, a route change can also cause a tunnel to completely disappear. If the tunnel agents detect that they are now connected by a continuous chain of RSVP routers, they inform the other tunnel agents along the path – if there are any – as well as the user agents on the application hosts and terminate.

5 Implementation issues

In this section some implementation details will be given related to the system of tunnel agents. In particular, we describe some implementation choices, the basic mechanisms for the migration of agents, the transfer of the execution state and the communication between agents. Further details can be found in [14]. Some specific aspects related to the interaction of tunnel agents to the RSVP protocol and the way in which a tunnel is characterized in terms of delay and other QoS parameters, are still being studied, and further implementation is required.

We have decided to use Java as the programming language for our agents, for its features of modularity and portability as well as for the possibility to dynamically load classes from different sources and for its object serialization capabilities [11]. We make extensive use of these basic mechanisms as they provide a valid support for the procedures of tunnel identification and tunnel agent migration inside the network. The remaining of the section covers agent mobility and communications issues, while some details on the environment to be installed in each router to allow the execution of tunnel agents are given in Section 5.3.

5.1 Implementing Agent Mobility

Mobility is the characteristic that enables tunnel agents and discovery agents to move from a router to another in order to achieve its goal. One feature provided is to allow an agent to move with its state, in order to resume the execution from the point where it was interrupted before the migration.

An agent in execution consists of: the code (*program state*), the content of the variables (*data state*), and the stack (*execution state*). In general, a complete migration requires the transfer of all of the three components of the state. Yet, even if program and data state can be managed with a limited effort, the capture, transfer and restoration of the execution state may cause considerable problems. In particular, in the case of an interpreted language such as Java, a portion of the execution state is included in the state of the interpreter, and the capture of such state, without modifying the interpreter, is practically impossible. The specific application in question gives the agents a high level of competence, so migration is never initiated from outside, but is started by the agent itself when some specific conditions occur.

For example, let us consider the mechanisms of identification of tunnels (described in Section 4.1) through discovery agents, and the migration of tunnel agents that have already been instantiated as a result of the relocation of the session path (described in Section 4.4). In both cases agent migration capabilities and status information are necessary: more specifically, a discovery agent has a list of routers already visited indicating with a flag whether the router supports RSVP capabilities; after a route reconfiguration, a tunnel agent needs information characterizing routers within the unchanged part of the original tunnel. Since the migration is performed under full responsability of the agent, the information needed for the correct resumption of execution can be saved within some variables of the object. As these variables belong to the data state, they can be transferred and later restored when the execution of the agent is resumed in the site of destination.

The mobility of tunnel and discovery agents is based on the Object Serialization facility present in Java version 1.1. Object Serialization is a mechanism that permits representation of an object as a stream of bytes. All the variables of the object are saved within the serialized representation of an object, together with all the references to objects contained in it.

When an object is serialized, Java does not store the bytecode of the class to which the object belongs in the byte stream. Thus, since in our system the classes needed are not always in the destination nodes, another characteristic of Java had to be exploited: the possibility to load some classes dynamically at runtime from different sources. This was done by using a NetworkClassLoader, whose task is to load the classes that are necessary for the execution of an agent. If the class needed is not present locally, the NetworkClassLoader searches for it within the nodes contained in a list specified during the configuration and the startup of routers, but that can be updated at runtime, thanks to some information brought by the agents. Such a mechanism permits therefore good exploitation of the network. The transfer of the bytecode of a class will occur only when required (in case of migration of several instances of the same agent, we do not need to transfer the same class); in any case, the transfer can always be done from the closest node, and not necessarily from the original node of the agent. This feature helps a lot once our system reaches a steady state condition, since the majority of the classes are already present on the routers and downloading actions do not occur very often. Furthermore, this mechanism avoids downloading huge amount of software on each router that might not be used at all. An extension to new types of agents is also easily supported: the classes the new agents may require are automatically downloaded, thus guaranteeing future extensability of the system.

5.2 Implementing Communication Mechanisms

The ability of an agent to communicate with other agents is another basic characteristic that is supported. In our system, the communication among the agents takes place through the exchange of messages that may be synchronous or asynchronous. In our opinion, this solution (unlike some mechanisms based on RPC) is very flexible and permits implementation of several schemes of communication and synchronization among agents. Our platform provides basic mechanisms that can be conveniently used to manage synchronization between agents, according to their requirements.

The encoding and transmission of messages takes place in the same way as the migration of agents. In fact, the mechanism selected for transferring messages is Object Serialization, in order to permit the sending of complex objects among different agents.

In the case of a synchronous message, a behavior similar to an RPC is obtained. The sending agent calls a communication primitive passing the message and the identifier of the destination agent as parameters and then it waits for a reply message. Conversely, in the case of an asynchronous message, the sending agent invokes another primitive and, after sending the message, continues with its execution.

Asynchronous communication is used for the interaction among tunnel agents. Once a tunnel has been identified, the tunnel agents on the edges communicate with each other in order to share information on the behaviour of the tunnel. Synchronous communication is used instead by a tunnel agent to retrieve information from the routers within the tunnel. This is implemented by sending a query to the local SNMP agent.

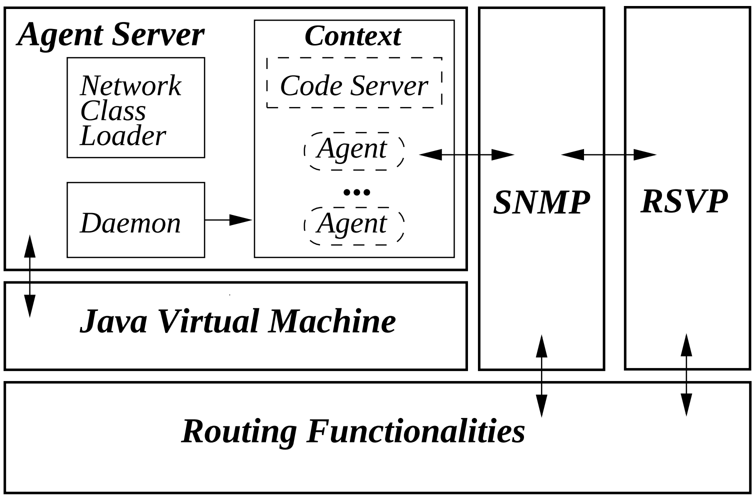

Figure 5: Software modules needed on a router

5.3 Agent Execution Environment

In this section we explicity focus on the software environment that has to be present on a router to allow agents' execution. For the sake of clarity, we also provide a full description of the reference architecture of our routers. Figure 5 shows the logical organization of a router main functional entities. With *Routing Functionalities* we indicate the basic mechanisms for packets routing, which depends on the router itself. They can be implemented directly in hardware or emulated through adequate software. In the current state of the implementation, routing functionalities are executed through a software approach; we use simple PC configured as routers, running Unix FreeBSD-2.2.5 [2].

An *SNMP agent* is required to monitor and retrieve information on the status of the router and on the different connections which go through the router. The *RSVP module* provides the router the capability to eventually use this protocol to communicate with other routers for resource reservation purposes. RSVP version 1, as implemented by the ISI rsvpd Release 4.2a1 [3], and the packet scheduler mechanism, provided by ALTQ version 0.4.1 [4], are currently used. Finally, an *Agent Server* is also instantiated on a router which represents the execution environment for the agents. As these agents are written in Java, a Java Virtual Machine is active on the routers. JDK 1.1 (Java Development Kit) for FreeBSD is currently used [5]. A specific analysis on the Agent Server's components is carried out in the following paragraphs according to the reference structure depicted in Figure 6.

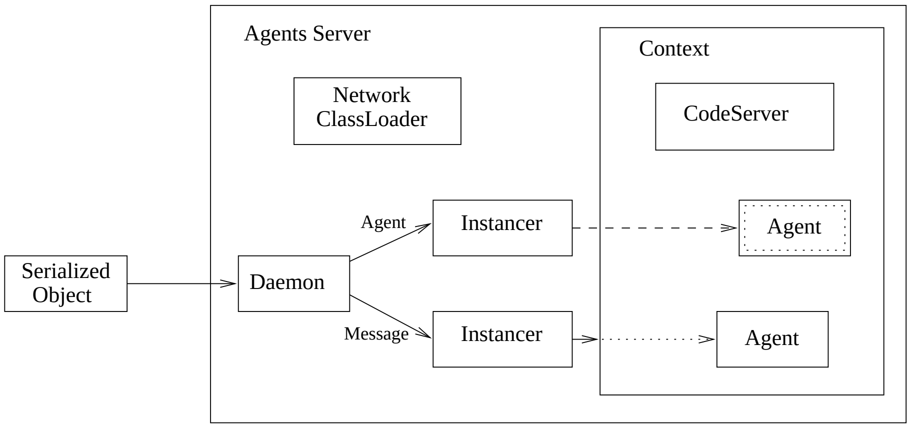

Figure 6: Agents Server

The *Daemon* is constantly listening to an input port waiting for tunnel or discovery agents

[2]FreeBSD is available at http://www.freebsd.org

[3]RSVP (ISI release) is available at http://www.isi.edu/rsvp/

[4]ALTQ is available at http://www.csl.sony.co.jp/person/kjc/software.html

[5]JDK 1.1.5 for FreeBSD is available at http://www.freebsd.org/java/

migrating from other nodes, or receiving messages to be delivered to local agents. Both messages and agents travel in a serialized form. Each time a stream arrives from the network, the Daemon creates a specific entity called *Instancer*, whose task is to instance the serialized object, both if it is an agent and a message. In both cases, the object, once it is instanced, is passed to the Context, whose task will be to make it run (if it is an agent), or to send it to the receiver agent (if it is a message). The Context is the object which knows all the agents currently instantiated on the node and provides all the functionalities needed for their management. The Context owns specific methods for creating, executing, suspending, deactivating, re-activating and killing an agent. Each serialized object arriving from the network is always passed to the Context. If this object is an agent, the Context initializes it, providing a reference to the Context itself: it then starts its execution and updates the list of the agents running locally. If the object is a message, the Context will deliver it to the destination agent, after having verified its availability to receive the message.

The *NetworkClassLoader* allows tunnel agents to execute on an Agent Server even if their class is not present there. When one of the classes the agent references is not present locally, the NetworkClassLoader downloads it from one of the Agent Servers indicated in a list managed by the Context. The class is then stored in the global cache of the Context: in such a way it is accessible also from the other agents currently executing on the server.

The Context contains also an object named *Code Server*: it is dynamically created each time a class is requested from a NetworkClassLoader. Its goal consists in making the requested classes accessible, if they are present locally. The Code Server contains a table where all the classes available on the node are listed.

5.4 Implementing QoS measurements at the tunnels by means of a flexible transport protocol

To enable the tunnel agents to carry out the QoS measurements, the transport protocol should include mechanisms that support this task. Therefore, a simple, yet flexible transport protocol for realtime user data has been designed and implemented. The PDUs of the *Flexible Continuous Media Transfer Protocol* can be configured to contain timestamps and/or redundant data like sequence numbers or checksums so that delays, loss and error rates can easily be monitored. Since these protocol mechanism can also be used for usual protocol functionalities, an integrated concept is being investigated. The design of the protocol, however, is beyond the scope of this paper.

6 Conclusions

We have presented an approach based on intelligent software agents for complementing currently still defective reservation schemes such as RSVP in the Internet. More complete QoS provisioning can thus be achieved in a flexible and highly scalable way. We believe that future telecommunication networks will be increasingly open so that services like QoS-management functionalities can be deliberately added. A Java-based agent platform seems to be particularly promising in a heterogeneous environment, which distributed multimedia systems are most likely to face. Further studies are needed to investigate the efficiency of our implementation. In particular, experiences will be necessary to determine agent-control strategies to avoid oscillation effects and to provide limits for a timely response of the agents.

Network Management

`(This text should be placed in a text box.)`

Network Management [16] can be defined as a collection of additional functionalities of a data telecommunication network that enables the management of that network. A *Network Management Protocol* like Internet's *Simple Network Management Protocol (SNMP)* is used to exchange the necessary control information to manage network operation aspects like failures, performance, configuration, security, and accounting. Management actions are carried out by control communication between a remote *Network Management station* (control console) and several *Network Management agents* that have direct access to the relevant software and hardware components of the network. Unlike the mobile agents discussed in detail in this paper, Network Management agents are hardware dependent, colocated with the components they make accessible, and thus immobile. To logically organize the managible information in a Network Management system, the managed objects are organized in a conceptual database termed *Management Information Base (MIB)* that is subdivided into individual MIBs for individual software / hardware components. With the abstraction of a conceptual database, network management actions can be interpreted as query and update operations in a distributed database system.

Quality of Service

`(This text should be placed in a text box.)`

With the deployment of multimedia services in data telecommunication networks the concept of **Quality of Service (QoS)** support has been introduced to data networking. Transmission of time-related data like digital audio and video demands stringent timeliness in the process of data transmission but usually allows occasional data loss or corruption. Therefore, in a network that supports QoS, a *QoS negotiaton* is part of the connection setup procedure. Quality of Service for data transmission services is usually expressed by *QoS parameters* like *data error rates, packet loss rates, throughput, end-to-end delay* and *delay variation (delay jitter)*. In a *QoS contract* between service user and service provider, the service provider commits itself to outperform the agreed-on limit given that the service user imposes a load not exeeding an agreed-on traffic specification. To achieve this goal, the service provider reserves a sufficient share of its resources for the connection under concern. Additionally, the user data traffic may be monitored and/or enforced to comply to the traffic specification. If for whatever reason the agreed-on Quality of Service cannot be achieved, an indication is given to the service user. A *self-adapting application* can use this indication to adapt its way of operation to the lower offered QoS by (e.g.) changing the way and/or extent the user data is compressed.

RSVP

`(This text should be placed in a text box.)`

For the support of QoS in the Internet, several approaches are currently discussed. The most prominent one is based on a framework usually referred to as **RSVP**, named after its central control protocol *Resource reSerVation Protocol (RSVP)* [18, 2]. This protocol is designed to set up and maintain information on the amount of necessary resource reservation along the path that will be taken by the user data packets. The RSVP protocol relies on standard Internet routing functionality to follow this route and interacts with all RSVP-enabled nodes along the path. Since the reservation of resources is a concept new to the Internet technology as used today, the deployment of RSVP demands for the extension of functionality of network nodes. Because of the enormous number of Internet nodes, RSVP was designed in a way that it can be executed even if legacy nodes without RSVP support are still present along the path. However, the effect of QoS support may be degraded by the uncontrolled behaviour of these nodes.

Java

`(This text should be placed in a text box.)`

Java is an object-oriented, portable, interpreted, multithreaded programming language developed by Sun [9]. It was devised for the development of safe, robust applications in heterogeneous network environments [11].

Independence from the underlying architecture is achieved by adopting a two-stage code generation process. The source code is compiled by a Java compiler which does not generate a *"machine code"* directly but a so-called *"bytecode"*, i.e. a high-level architecture-independent code for a hypothetical machine implemented by the Java interpreter and its runtime system, indicated as a *Java Virtual Machine (JVM)*. The bytecode can then be executed on any machine with a Java interpreter. As the JVM was produced for a number of platforms right from the first version of the language, this ensures that a Java application can be executed on any platform on the market. Architecture-independence is not only useful for applications to be used in networks, but also for problems relating to the distribution of software on different platforms: a programmer no longer has to worry about developing various versions for different machines and operating systems.

Modern network-based applications typically need to do several things at the same time. The Java *multithreading* capability provides the means to build applications with many concurrent threads of activity. Although recent operating systems provide libraries for multithreading, their difficulty and dependence on a particular architecture have prevented them from being used by a large number of programmers. Java, on the other hand, supports multithreading at the language level, providing sophisticated synchronization primitives based on the concepts of *monitor* and *condition variables*.

A basic concept of the Java architecture is that of *class loaders*: they allow the Java runtime system to load classes without any knowledge of the underlying file system. All Java Virtual Machines include a default class loader which loads the classes of the local file system, but Java allows the user to define his own class loader, thus overriding the behaviour of the default loader. This makes it possible to load classes from various sources, as is done, for example, in applets, which load classes from a Web server using the http protocol.

Java 1.1 has introduced an interesting feature called *object serialization* which allows to take an object and write its state to a byte stream. At a later time, the object may be restored deserializing the sequence of bytes previously saved. By means of this mechanism, objects persistence can be implemented : an object's lifetime is not determined by wheter a program is executing because it can lives between different invocations of the program.

References

[1] F. Baker, J. Krawczyk, and A. Sastry. RSVP Management Information Base. *RFC 2206*, September 1997.

[2] R. Braden, L. Zhang, S. Berson, and S. Herzogand S. Jamin. Resource ReSerVation Protocol (RSVP) – version 1 Functional Specification. *RFC 2205*, September 1997.

[3] D.M. Chess. Mobile Agents: Are they a Good Idea. *IBM Technical Report RC19887*, October 1994.

[4] H. de Meer, A. Puliafito, and O. Tomarchio. Management of QoS with Software Agents. *Cybernetics and Systems: An International Journal*, 27(5), 1998.

[5] O. Etzioni and D.S. Weld. Intelligent agents on the Internet: Fact, Fiction, and Forecast. *IEEE Expert*, 10(4):44–49, August 1995.

[6] S. Fischer and H de Meer. Decision Support in Cooperative QoS Management. In *Proc. of the 6th Int. Workshop on Quality of Service (IWQoS'98)*, San Francisco, USA, May 1998.

[7] Michael R. Genesereth and Steven P. Ketchpel. Software agents. *Communications of the ACM*, 37(7):48–53, July 1994.

[8] G. Goldszmidt, Y. Yemini, K. Meyer, M. Erlinger, J. Betser, and C. Sunshine. Decentralizing Control and Intelligence in Network Management. *Proc. of the 4th International Symposium on Integrated Network Management*, May 1995.

[9] J. Gosling. The Java Language Environment: a White Paper. Technical report, Sun Microsystems, May 1995.

[10] A. Hafid and G.v. Bochmann. Quality of service adaptation in distributed multimedia applications. *ACM Multimedia Systems Journal*, 6, 1998.

[11] M. A. Hamilton. Java and the Shift to Net-Centric Computing. *IEEE Computer*, 29(8):31–39, August 1996.

[12] A. Lazar. Programming telecommunication networks. In *Proc. of the 5th Int. Workshop on Quality of Service (IWQoS'97)*, pages 3–22, New York, May 1997.

[13] A. Puliafito, O. Tomarchio, and H. de Meer. An agent-based framework for QoS management. In *First World Congress on System Simulation WCSS'97: 4th ANMT on QoS*, Singapore, September 1997.

[14] A. Puliafito, O. Tomarchio, and L. Vita. A Java-based Distributed Network Management Architecture. *Third International Conference on Computer Science and Informatics, CS&I'97*, March 1997.

[15] S. Shenker, C. Patridge, and R. Guerin. Specification of Guaranteed Quality of Service. *RFC 2212*, September 1997.

[16] W. Stallings. *SNMP, SNMPv2, and CMIP. The practical guide to network-management standards.* Addison Wesley, 1993.

[17] J. Wroclawski. Specification Controlled-Load Network Element Service. *RFC 2211*, September 1997.

[18] L. Zhang, S. Deering, D. Estrin, S. Shenker, and D. Zappala. RSVP: A New Resource Reservation Protocol. *IEEE Network*, 7(5):8–18, September 1993.