Presented at *12th International Conference On Computer Communication, August 21-24, 1995, Seoul, Korea*

Relationships among Inter-Dependent Real-Time Streams

Luca Delgrossi, Sibylle Schaller, Lars Wolf

IBM European Networking Center, Vangerowstraße 18 • 69115 Heidelberg • Germany

Phone: +49-6221-59-3000 • Fax: +49-6221-59-3400

{luca, schaller, lars}@heidelbg.ibm.com

Abstract: Multicast real-time streams assume in general the shape of a tree where the source is the root and the receivers are the leaves. Many common networked multimedia applications need to create a large number of such streams to connect the end-peers one to each other. Often, relationships among the streams created by the same or related applications exist. Being able to express these relationships makes it possible to optimize the global utilization of the network resources. This paper suggests the concept of groups of streams to express such relationships and defines a set of basic relationships that are felt of immediate importance for emerging networked multimedia applications. The way global network resources can be optimally exploited is also presented. The results can be easily applied to existing setup protocols.

Keywords: real-time stream, quality of service, group, relationship, resource sharing, setup protocol.

1 Introduction

Distributed multimedia applications exploit continuous media data streams to deliver digital audio and video information over the network. In this communication scenario, the transport system has to provide certain real-time guarantees to such streams in order to serve the applications with the required quality of service. There is also a requirement for multicast capabilities, i.e. the streams have to be able to convey the data simultaneously to multiple destinations. In general, the applications need to create and make use of a fairly large number of these multicast real-time streams: in a video conferencing application, for instance, each speaker has to be connected to all the participants.

Often, there exist inherent relationships among real-time streams. For instance, it is in most cases useless to receive a video stream unless the correspondent audio stream is also present: think of a movie without its soundtrack or of a tele-learning application where it is possible to see the teacher but not to hear his lecture. Further kinds of relationship exist other than this basic one: a common example is inter-stream synchronization or, again, a set of streams sharing certain resources, e.g., bandwidth over a network link or multicast groups addresses.

Applications are usually aware of these relationships among streams but they do not have the means to exploit this knowledge. Often, the issues have to be solved by the transport provider, i.e. at the transport or network layer. Consider a router that performs bandwidth management of the outgoing links and is traversed by two real-time streams. If the router knew that only one of the two streams can be active at a time, it could optimize bandwidth usage by allocating only the sufficient bandwidth for a single stream and having the two streams use it in turn.

To allow applications to express inter-stream relationships, this paper introduces the concept of *groups* of real-time streams. It defines an abstraction for groups and subgroups and it analyses relationships among streams belonging to the same group and the ways these relationships can be expressed. Finally, it considers how the information about groups can be efficiently distributed in the network and it discusses the impact on current setup protocols and resource management subsystems.

Being able to express the relationships among inter-dependent real-time streams and to communicate this information to the transport system is felt to be important because:

- *it facilitates structuring large applications*, e.g, a conference or a seminar, into several logical modules: in a conference, for instance, different groups may correspond to the diverse roles of the participants – listeners, speakers, and interpreters.
- *it helps identifying related streams*: this can be used in many different ways, e.g., to apply the same charging to the group members or as a hint on which streams to delete when stream preemption becomes necessary.
- *resources can be allocated on a per group basis* instead of on a per stream basis: since not all members of a group are simultaneously active, this leads to a significant reduction of the global resource reservation.
- *it improves the service provided by the network*: for instance, methods are required to mark streams that should be synchronized on playout at the receiver.

This paper is organized as follows. Section 2 briefly summarizes related work in the literature. The group and relationship abstraction is presented in Section 3. Issues related to group management and the distribution of group information in the network are presented in Section 4. A description of an implementation design for the group abstraction is provided in Section 5. Finally, the last section concludes the paper by summarizing the results.

2 Related Work

So far, the concept of groups of streams has not been fully exploited. Although some work has been done with the main objectives of reaching a higher degree of resource sharing, today it is still at a preliminary stage. No definitions of the relevant inter-stream relationships have been agreed on nor implementation experience is available. Each application handles its streams independently and yet there are no means to exploit common characteristics of a set of streams.

Gupta and Moran present their abstractions for groups of channels in [GuMo93] as part of their design of the Tenet Protocols Suite. They introduce three basic relationships and discuss how they can be used by different conferencing applications. Their work has been taken as the basis for this discussion.

The original specification of the ST2 protocol [Topo90] includes the definition of groups of streams, but in an incomplete form. The RSVP protocol [ZDES93] uses *reservation filters* to minimize bandwidth usage in multicast groups. The filter concept allows heterogeneous receivers to share resource reservations on a per sender basis. Reservation filters also allow switching among several incoming channels. This approach addresses the optimal bandwidth usage problem, but it does not contain a general mechanism for groups of streams/channels, i.e. the ability to exploit other relationships.

In [MaCG94], concepts of group communication for the management of multipeer communications in the transport layer are introduced. Groups are defined as objects by type (set of rules) and by an instance (set of members) with one or more associations between group members. [Bult93] introduces a formalism to coordinate such activities of multimedia applications as managing resources, synchronization and interaction.

The approach presented in this paper deepens the discussion on groups of real-time streams. Groups, subgroups, and relationships are formally defined and a richer set of relationships is presented. Also, the intuitive concept of relationship orthogonality is introduced. Finally, we analyse how support for the most relevant relationships can be embedded in a real multimedia transport system. Ideas included in this paper served as the basis for the design of the group mechanism of the most recent version of the ST2 protocol, ST2Plus, recently defined by the IETF ST Working Group [Delg94].

3 Groups

This section defines groups and subgroups of real-time streams and a number of inter-stream relationships.

3.1 Definition of Groups

A group G consists of a set of streams S and a relationship R. For each couple of different streams belonging to S, the relationship R is valid, i.e. every member is in relationship with each of the others. This can be formally expressed in the following way:

If a stream α is in relationship R with a stream β, we write:

$$(\alpha\, R\, \beta)$$

Let now S be a set of streams:

$$S = \{s_1, \dots, s_N\}$$

so that,

$$\forall\, i, j \quad (s_i\, R\, s_j) \text{ with } 1 \leq i, j \leq N;\ s_i, s_j \in S$$

Then, we define a group G as:

$$G = (S, R)$$

By definition, it is allowed for the set S to be empty: G = (Ø, R). This is of course a special case: working with groups makes real sense only if the set S has at least two members. Empty groups may exist, e.g., during the establishment phase of an application if the stream members in the group are not yet established. The relationship R holds between members of the group as soon as they are added to the set S.

3.2 Subgroups

It is also possible to define the sum operation among groups. Consider group $G_1 = (S_1, R_1)$ and $G_2 = (S_2, R_2)$. Given a new relationship R, we define G = (S, R) as the *sum* of G_1 and G_2 when:

(a) R holds for S, where S includes all members of S_1 and S_2.

(b) R_1 (R_2) holds for members of S_1 (S_2).

(c) R is *orthogonal* with respect to R_1 (R_2).

In other words, G_1 and G_2 are subgroups of G. Their members are subject to the R_1 (R_2) relationship and to the relationship R holding for group G.

The concept of *orthogonality* is presented in an intuitive way: a relationship is orthogonal with respect to another when there are no conflicts between the two, so that it is possible that they both hold at the same time for the same streams. Relationships as “R_x: all streams follow the same routes” and “R_y: all streams go over disjoint paths” are clearly not orthogonal and it is not possible to apply the sum operation.

3.3 Inter-Stream Relationships

Defining appropriate inter-stream relationships is the key to the full exploitation of the group of streams abstraction. Relationships express the common properties among streams: they may indicate that the streams share some communication objects or that they are somehow related. In the following, the relationships that are felt of immediate importance for emerging distributed multimedia applications are presented.

Bandwidth Sharing

Several inter-stream relationships are based on a single concept: resource sharing. This is useful because it allows to associate independent streams with a single resource reservation. The *bandwidth sharing* relationship is certainly the most common, and with the development of setup protocols, this relationship will be of fundamental importance. For instance, RSVP already has some built-in features for bandwidth sharing.

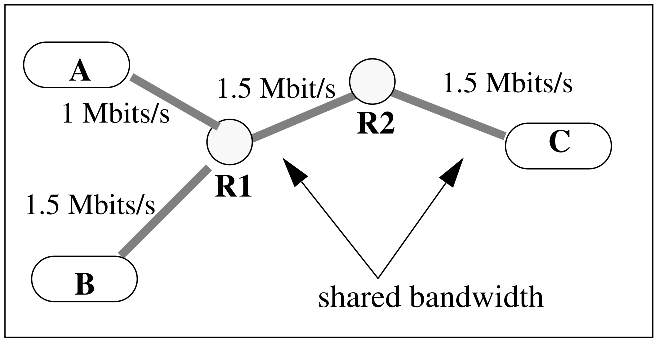

Figure 1: Bandwidth Sharing

A video conferencing application may exploit the bandwidth sharing relationship by allocating sufficient bandwidth for just one speaker and let in turn the current speaker use it. The alternative to allocate bandwidth on a per-speaker basis results in overbooking of resources. In Figure 1, 1.5 Mbits/s are reserved between R1 and C instead of 2.5 Mbits/s. We discuss group bandwidth management in Section 5.2.

Fate Sharing

In a group of streams, it is possible that the streams depend on the existence of one or more "master" streams. The *fate sharing* relationship indicates that all streams in the group should be deleted if one of the masters is deleted. It mainly applies to situations where the streams are inter-dependent and it does not make sense to run them if the "master" stream is not present. For instance, the fate sharing relationship could be exploited by a remote training application in which an instructor shows how to fix a failed hardware component. Just as it is not useful to watch the instructor without hearing his comments, it is not sufficient to listen to him without being able to observe how the fixes have to be done. In this case, the audio and video streams originated by the instructor are in a fate sharing relationship. A further example is provided in Section 3.4.

Fair Preemption

If a stream with high precedence has to be run and the available capacity is insufficient, a possibility is to preempt one or more low precedence streams so that the associated resources are freed. The choice of which low precedence streams to preempt is critical: information is required to minimize the number of applications affected by the interruption. By specifying that the *fair preemption* relationship exists among the streams, an application indicates that its streams are related together and, if one of them gets preempted, then it would be a fair choice to preempt the others as well. This helps freeing resources for usage by high precedence streams without disrupting other applications.

Path Sharing

Path sharing can be used to increase the efficiency of other relationships. For instance, the most benefits from bandwidth sharing are obtained when the streams follow the same network paths. Path sharing can also be used to convey the streams through convenient high-speed links or to avoid that the data flows through a given node, e.g. for security reasons. This can be implemented by first building a stream with a route specified by the source and then having the other streams share the same paths with it. The left side of Figure 2 serves as an example for path sharing.

Disjoint Paths

The disjoint paths relationship is exactly the opposite as path sharing (Figure 2, top). It serves the purpose of conveying related streams over different network paths. One reason for doing this could be for instance the desire to provide reliable communications: in this case the data is duplicated and sent over different paths so that if the first path fails, the data will still reach its destinations through the second path.

Related QoS

It may be convenient to specify that some streams have related needs in terms of Quality of Service (QoS). Usually, this applies to streams originating from the same source and directed to the same targets. For instance, it could be specified that the data belonging to stream S2 reaches the destinations within 25

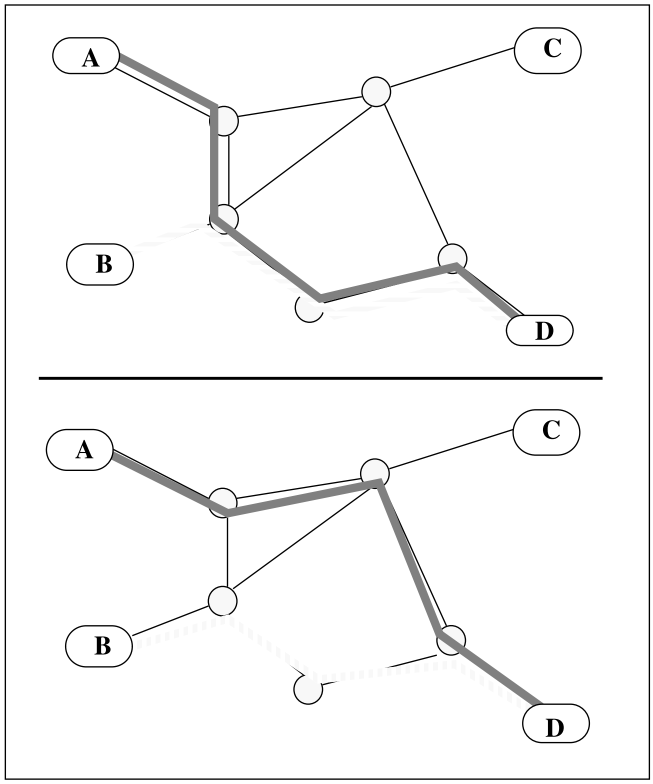

Figure 2: Path Sharing (above); Disjoint Paths (below)

msecs after stream S1 data. This imposes a limit on inter-stream delay jitter and implies being able to express a relationship between two separate QoS. The *related QoS* relationship can be used when information belonging to a single logical stream is carried over separate streams at the network layer. For instance, it may be necessary to send hierarchically encoded video as MPEG-2 video over different streams and re-synchronize the data at the receivers.

3.4 An Example

In a distributed game application, the master controls 3 players, cf. Figure 3. The master delivers in real-time information related to the game contents to all players. Each player in turn updates the master on its actions and moves in the game scenario. The game fails if the master is not present and active.

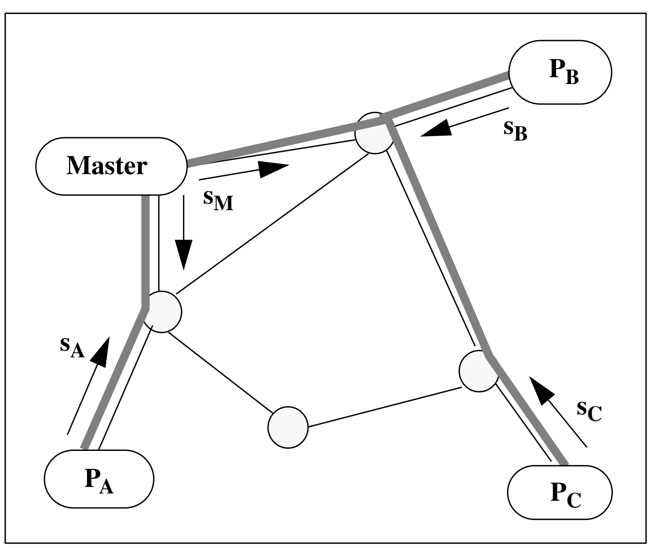

Figure 3: Sample Application Scenario

The following groups and relationships may be defined to model this application:

$G_1 = (S_1, R_1)$

$S_1 = \{s_B, s_C\}$ R_1 = bandwidth sharing

$G_2 = (S_2, R_2)$

$S_2 = \{s_M, s_A, s_B, s_C\}$ R_2 = share fate of s_M

Note that R_1 and R_2 are orthogonal.

4 Groups Management

The group and relationship abstractions have been so far presented from a general point of view. However, our interest is to analyse groups of multicast real-time streams. When the networking aspects are considered, several specific issues are raised on the most convenient way of managing groups. They are dealt with in this section.

4.1 Naming

Globally unique group names are required to univocally identify groups in the network. A group name space has to be defined. The way names are defined depends on the domain and on the implementation. In the Internet domain, a possible way to define a valid name space is to use a triple including a local identifier, the IP address of the local host, and a timestamp. This way, each host may generate group names univocally. An alternative is to let a central host act as "group name generator". Once a group has been assigned a name, the name can be used to add streams to it.

4.2 Membership

Several issues are related to group membership. Access to a group may be *open* or *restricted*. If the access is open any stream may become a member of the group; if the access is restricted, only a pre-defined set of streams is allowed to be member in the group. The latter can be achieved, for instance, if the complete list of permitted streams is made available: when a new stream requests to join the group, the list is checked for its stream identifier. Open groups can be used for public discussions or lectures at the university, while restricted groups could be used for managers meetings to discuss the company's future strategy.

Group membership can be static or dynamic. A static member is assigned to the group at creation time and it belongs to the group for its whole lifetime. A dynamic member may leave the group at any time and possibly join other groups. While static membership is easier to implement, dynamic membership allows for more sophisticated exploitation of the group abstraction.

Allowing for a single stream to belong to multiple groups is convenient, provided that the rule of relationships orthogonality is observed.

4.3 Maintaining Group Information

The amount of group information to be distributed over the network can be quite large. To perform *global* functions, it is required that all nodes traversed by at least one of the streams belonging to the group store information on all the group's streams. For instance, consider the relationship "R means: group membership is limited to 10 streams": each node needs to know how many streams currently belong to the group and, in particular, knowledge is requested on the streams that do not traverse the node itself.

Maintaining global knowledge to allow this kind of operations is probably too expensive. A reasonable solution seems to be maintaining at every node group information on the streams that traverse the node. Since stream information is collected during the stream setup phase, group information can be also distributed at the same time. The idea is to include group information into stream setup information, so that every node traversed by the stream receives it. Although this has the effect of limiting the amount of feasible group functions, it simplifies the design.

4.4 Security

Some security issues have to be discussed in the context of group management. This does not comprise data security: being member in a group does not automatically provide access to the data carried by other member streams, i.e. data security remains in the responsibility of the single peers. Rather, some measures have to be taken to protect group members from non-members streams that attempt to exploit the group relationship.

If a malicious stream has knowledge of the group name and which resources are used, it could try to become a member in the group and then to use up the shared resources without obeying the groups communication rules, e.g. one speaker at a time. Open groups and dynamic groups are clearly more vulnerable. Note, however, that not all relationship are subject to parasitism: a non-member stream joining a disjoint-path group hardly causes any harm to the group members.

4.5 Relationship Issues

The *combination* of basic relationships often allows to increase the efficiency: for instance, the overall minimum bandwidth usage is obtained by coupling the bandwidth sharing and path sharing relationships.

Some relationships require only *local knowledge*, while others need *global knowledge*. Local knowledge is sufficient to implement, e.g., the disjoint paths relationship: given a stream in the group, the routing algorithm tries to convey the next stream over a different path out of the node. Global knowledge is required to implement, e.g., fate sharing: if a stream in the group gets deleted the others are also dropped (global knowledge is required because the stream may be unknown by some nodes).

Some relationships play a role during the *stream establishment* phase only, some others affect *data transfer*: in a real-time communication environment, routes are often "pinned", i.e. they are selected at stream establishment and do not change over time unless failures are detected; on the contrary, special actions may have to be taken during the data transfer to ensure that the packet arrival delay between two related streams is inferior to a given time unit.

It is often easier to express relationships among streams *emitted by the same origin*: for instance, when a source emits audio and video as two separate streams or, when a source emits a hierarchically encoded multimedia stream comprising a bunch of logical substreams.

5 Implementation Design

To experiment with the group abstraction, it has been decided to extend the IBM ENC ST-II implementation. The following choices have been taken in the design:

The group name is a structure including:

```
struct groupname {
    short   local_id;      /* locally unique identifier */
    long    IP_address;    /* used as globally unique id */
    long    timestamp;     /* group creation timestamp */
};
```

This way, any ST agents may independently generate globally unique identifiers; local ids are assigned by an internal function that keeps track of the ids already in use. The groupname structure is included into the Group parameter and added to the CONNECT message used to establish a new ST-II stream. The Group parameter also contains information on the relationship. This way, group information gets distributed to all the nodes traversed by the stream.

To keep the design simple, only static membership has been foreseen. This means, the initial CONNECT message must contain information on all the groups of which the stream is a member. The stream leaves the groups only at DISCONNECT time.

An ST agent receiving the CONNECT message extracts the Group parameter when present and creates the correspondent entry into its local group database. The group database is maintained parallel to the normal ST-II database that contains stream information. The entry in the group database contains a pointer to the stream descriptor of the first stream member in the group. The other members are accessed via a regular pointers list. The group database entry also includes pointers to a special area. The contents of this special area depend on the group relationship.

It was decided to focus on the following relationships:

a) *path sharing*: the Group parameter contains a constant to indicate path sharing.

b) bandwidth sharing: the Group parameter contains a constant to indicate bandwidth sharing. A parameter specifies how much bandwidth needs to be allocated for the group.

The implementation of these two functions has an impact on the routing algorithm and local resource manager. This is discussed in the next two sections.

5.1 QoS Routing

To satisfy group management requirements, the routing algorithm has to be aware of inter-stream relationships. Our QoS-based routing algorithm, which is described in [VWHK95], uses metrics based on bandwidth, delay, and reliability. It choses the link which matches the specified requirements best.

Considering the inter-stream relations discussed in Section 3.3, bandwidth sharing, path sharing, disjoint paths, and related QoS have an impact on the QoS-based routing algorithm. The modifications required to implement group support include:

- a group identifier is passed to the routing algorithm,
- a flag indicates whether joint or disjoint paths should be selected,
- a flag indicates whether bandwidth is shared

Assume shared paths have been requested. The algorithm attempts to direct the flow along already established routes for the group whenever possible. It checks whether the QoS requirements can be satisfied, e.g., sufficient bandwidth is available or shared bandwidth for the group has already been allocated. Failures results in a different route being selected, but never affect stream creation.

5.2 Resource Management

With respect to groups of streams the following two questions about resource reservation are relevant: 1) when is the reservation done, and 2) which amount of bandwidth must be reserved. Both have an impact on resource utilization and whether more streams/groups get a chance to get established.

There are two ways of reserving:

- *At Once*: make all necessary reservation at once. Here, the first stream to cause reservations must know in advance the required amount for the group and how many streams will participate in the group. This approach may lead to over-reservation if not all expected streams join the group.
- *Stepwise*: at group creation time, only the actual needed resources are reserved. Additional necessary resources are reserved when new streams join the group. In contrast to the previous approach, this reservation mechanism is based on up-to-date information available at the reserving sites.

The amount of bandwidth to be reserved depends on the stream and application nature. Consider an application with N peers. Assume only k peers (with $1 \leq k \leq N$) may transmit data at the same time. If all streams are homogeneous, i.e. they require the same amount of bandwidth B, the maximum required bandwidth for the group is:

$B_G max = k*B \qquad 1 \leq k \leq N$

The maximum required amount is reserved *k* times (with *k* the number of streams sharing this resource). This causes waste if less than *k* streams run through the same path. The advantage of this method lies in its simple calculation.

Heterogeneous streams require each a different amount of bandwidth:

$b_1, b_2, \ldots, b_N$

Assuming streams are ordered by bandwidth needs, i.e. assuming:

$b_1 > b_2 > \ldots > b_N$

the maximum amount of bandwidth required for the group is:

$B_G max = b_1 + b_2 + \ldots + b_k$

Also, for nodes not traversed by s_1 (the stream requiring b_1):

$B_G max = b_2 + b_3 + \ldots + b_k + b_{k+1}$

In this case, only the *k* highest requirements are reserved. This allows better adaptation to less than *k* streams per path, because reservations are done only for the existing streams, but at most for those *k* streams. This calculation is more complex than the above, but it scales better to the real needs.

Additional parameters to be passed to the local resource manager include the group identifier, the value for “k”, and a flag to indicate homogeneous streams. Which of the four approaches will be applied depends on the flow specification and on the reservation mechanism.

6 Conclusion

Expressing inter-stream relationships allows the network to improve the services it provides and to make an optimal use of the available resources. We introduced the group abstraction to express these relationships. We identified a set of fundamental relations and showed how to embed group information in the routers during stream setup. As the next step, we feel that it is necessary to design the appropriate mechanisms to support streams with related QoS including inter-stream delay jitter.

References

[Bult93]
D.C.A. Bultermann: *Specification and support of adaptable networked multimedia*, ACM Multimedia Systems, Vol 1, pp:68-76, 1993.

[Delg94]
L. Delgrossi (Ed.): *Internet Stream Protocol, Version ST2Plus*, IETF ST Working Group Internet-Draft, work in progress, October 1994.

[GuMo93]
A. Gupta, M. Moran: *Channel Groups - A Unifying Abstraction for Specifying Inter-stream Relationships*, Tenet Group, UCB and ICSI Berkeley, TR-93-015, March 1993.

[MaCG94]
H. Maisonniaux, P. Cocquet, M. Gagnaire: *New Concepts for Multipeer Communications*, Proceedings of 4th Open Workshop on High Speed Networks, Brest, 1994.

[Topo90]
C. Topolcic: *Experimental Internet STream Protocol, Version 2 (ST-II)*, RFC 1190, October 1990.

[VWHK95]
R. Vogel, H. Wittig, R. G. Herrtwich, W. Kalfa, L. C. Wolf: *QoSFinder: A Method for the Routing of Multimedia Streams in Computer Networks (in german)*, KiVS 1995.

[ZDES93]
L. Zhang, S. Deering, D. Estrin, S. Shenker, D. Zappala: *RSVP: A New Resource ReSerVation Protocol*, IEEE Network, 1993.