Internet Engineering Task Force
INTERNET DRAFT

D.-H. Gan/R. Guerin/S. Kamat
Juniper Networks, Inc./IBM/IBM
T. Li/E. Rosen
Juniper Networks, Inc./Cisco

21 November 1997

Setting up Reservations on Explicit Paths using RSVP
draft-guerin-expl-path-rsvp-01.txt

Status of This Memo

Abstract

This document presents motivations for extensions to RSVP in order to enable setting up of reservations on explicit routes. The advantages of providing this support are discussed in the context of MPLS and QoS routing. An approach to providing these extensions by means of opaque routing objects in RSVP messages is presented.

Contents

1. Introduction

The purpose of this document is to introduce and motivate extensions to RSVP to enable setting up of reservations on explicit routes. Enabling reservations on explicit routes can be useful in several different contexts. In particular, it can be used to ensure that certain flows use a ``label switched'' path as in the MPLS context [CDF+97] or to facilitate the management of QoS paths computed by a QoS capable router as in [GKO+97]. In this document, we describe further these potential benefits, and show how they can be attained with minimal impact to RSVP. It should be pointed out that the focus of this document is on unicast flows as there are many other issues that need to be addressed to consider the use of explicit routes for multicast flows.

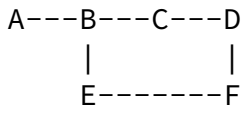
In the context of unicast flows, explicit routes are to be specified through a new Explicit_Route object in RSVP. This object, like policy objects, is opaque to RSVP which only needs to ensure its delivery to routing. Routing is responsible for processing the Explicit_Route object, and will use the information it contains to construct its response to a Route_Query from RSVP.

Sections 2 and 3 motivate the need for explicit route support within RSVP in the context of MPLS and QoS routing respectively. Section 4 describes the specific mechanism of setting up reservations on explicit paths. This includes specification of a format for the

Explicit_Route object and the interactions between RSVP and routing in this context.

2. Bandwidth Reservation for Explicit Route in an MPLS Environment

Consider the following topology:



Let us suppose that this topology exists in the network of an Internet Service Provider (ISP). We suppose further that node A has an interface to one of the ISP's subscribers, S1, and node B has an interface to a different subscriber, S2. Finally, we suppose that both subscribers are generating packets that are addressed to destinations reachable only through node D.

In order to make the best provisioning of its bandwidth, the ISP may decide that such packets from S1 should follow the route A-B-E-F-D, while such packets from S2 should follow the route B-C-D. Further, the ISP may want to reserve resources for each of these "flows", so that it can schedule transmissions along the respective routes in a way that corresponds to whatever agreements the ISP has made to the particular subscribers.

Putting this decision into effect in a conventional IP network is extremely difficult, since it requires that two packets going through B, with the same destination, be sent on separate routes. Therefore, ISPs tend to use ATM or Frame Relay networks to provide this level of bandwidth management. ATM and Frame Relay networks also provide the capability to support whatever resources reservations are necessary.

MPLS [CDF+97] provides a way for an ISP to obtain this functionality without the need to resort to ATM or Frame Relay. In MPLS, node A can apply a "label" to packets from S1 which must pass through D; node B can apply a label to packets from S2 which must pass through D. When a labeled packet is transmitted, the label is sent along with it. Once a packet is labeled, the forwarding decision is based only on the label, NOT on the contents of the packet header. Thus there is nothing to prevent packet P1 from traveling the A-B-E-F-D path, while packet P2 travels the B-C-D path, even if P1 and P2 happen to have the very same destination address.

Of course, MPLS must incorporate some "path setup" procedure whereby paths that differ from the "normal" IP routing can get explicitly set up. MPLS must also incorporate some means of performing resource reservation along these paths. While a resource reservation protocol could be designed exclusively for MPLS, it would seem to make most sense to use RSVP for that purpose; after all, RSVP was designed to be the resource reservation protocol of the internet.

This requires some modification of RSVP. As currently specified, there is no way to force an RSVP Path message to follow any path other than the "normal" path to a particular destination. So if a different MPLS path were set up for certain flows, there is currently no way to get the Path message to follow that path.

If RSVP control messages could carry opaque objects that are meaningful to routing and RSVP's interface to routing is broadened as in [GKR97] so that RSVP could pass such objects to routing, then this difficulty can be overcome. The Path messages could carry an explicit route object. To determine the next hop for the flow, RSVP would pass the Explicit Route Object (and other opaque objects if present) to routing, which would pass back the identity of the next hop, and a modified Explicit Route Object. This would force the Path

message to follow the path of the corresponding MPLS flow and ensure that resources are reserved along the MPLS path.

The general ability to carry an opaque routing object in RSVP messages further enables one to combine the setup of an MPLS path with resource reservation along the same path. This could be achieved by having a second opaque routing object carry an MPLS flow identifier (label) in conjunction with the explicit route object. The definition of such an MPLS label object is deferred to another draft. Clearly, this approach has the advantage of avoiding a second round trip to make reservations along the MPLS path when the path set up itself must be done first. The need to have a second round trip seems to simply add latency and complexity, without adding any value.

3. QoS Routing with Explicit Routes

An objective of QoS routing is to choose for each flow the path that has the best likelihood of meeting the flow's QoS requirements,

while still making efficient use of network resources. In order to achieve this goal, QoS routing requires knowledge of network resource availability and of the QoS requirements of the flows. This information can be provided in a number of ways (e.g., see [CNRS97, GKR97] for possible approaches) and is then used by a QoS path selection algorithm to identify an appropriate path for a flow. The selection of a path and the distribution of the information needed to make that selection, however, only represent a subset of the functions needed to support QoS routing. There are two other important issues that a QoS routing solution must address to meet its objectives. These are:

1. Management of QoS paths of individual flows, and

2. Enforcing high level admission control policies.

Management of QoS paths includes not only setting up the paths correctly, but also maintaining or adjusting them in response to failures and changes in the network. High level (call) admission policies are needed (see [CNRS97] for a discussion of this issue) to control how selected paths are being used so as to preserve the long term efficiency of the network. For example, a suitable path might be found for a flow, but rejected by the high level admission control because of its cost to the network, e.g., it is using a large number of links which could alternatively be used to support several such other calls to different destinations.

In the rest of this section, we articulate how explicit routes can facilitate handling of these two issues. However, before doing

so, we briefly compare, in the context of QoS routing, the use of explicit routes versus the hop-by-hop routing approach presented in [GKO+97].

A hop-by-hop routing solution has the benefits of requiring the least changes to RSVP and possibly offering added flexibility (see [GKH97] for details), but this does come at a cost. Specifically, with hop-by-hop routing, there are multiple decision points (each hop) involved in selecting a path, with each making independent decisions. As a result, end-to-end control of a path requires coordination between the multiple decision points, and this can often be a complex task. For example, even in the context of a link

state routing protocol such as OSPF where all routers in a domain compute their routes using the same algorithm applied to a common topology database, no single router has complete knowledge of the actual path being followed. This is because inconsistencies during routing transients as well as equal cost multi-path considerations, independently affect local path selection decisions. Additional mechanisms are, therefore, needed to coordinate these independent decisions.

On the other hand, when explicit routes are used, selection of the entire path is made at a single decision point (the first router in the path). In the rest of this section, we expand on the benefits of a single decision point in the context of both QoS path management and high level call admission.

3.1. QoS path management

In best-effort routing, route changes occur relatively infrequently, and mostly when local interfaces change state or when routing updates are received from the routing protocol. With QoS routing, changes that would result in the selection of a new route for a given destination and QoS requirements are much more frequent, as they typically occur each time a metric update is received. If such changes were to trigger re-routing of existing QoS flows, this would translate into disruption of service to already established flows. Furthermore, this could also increase routing instability as such re-routing may trigger additional metric updates and cause further re-routing. Keeping a flow's routing state, i.e., the path on which it has established a reservation, ``pinned'' as long as the path remains satisfactory for the flow (and the network) is one possible approach to this problem. Path pinning, however, has a number of implications for QoS routing.

First, path pinning requires knowledge that the path being pinned is adequate. This includes several aspects. First and foremost, the

pinned path should be loop free. When an explicit route is used, this is readily achieved as the node selecting the explicit route can ensure it is free of any loop. In contrast, when hop-by-hop routing is used, the coordination of the multiple decision points involved in the selection of the path requires not only that all nodes rely on

the same routing algorithm, but also imposes close coupling with RSVP states to detect the formation of loops (see [GKH97] for details). Such a coupling adds some complexity, but more importantly, it can prevent the flow of data on the pinned path until after resources have been successfully reserved on the entire path (see again [GKH97] for details). In the case where reservations are successful on only a portion of the path, this means that the data may not be able to take advantage of such partial reservations. This is obviously undesirable, and while this can possibly be remedied (see also [GKH97] for possible approaches), solutions come at the cost of added signaling and processing complexity.

Besides being loop free, a pinned path must also be capable of satisfying the QoS requirements of the flow. Hence, it is important either to ensure the availability of resources on a pinned path, or to provide simple mechanisms to unpin it in case the required resources are not available when they are being requested, e.g., when an RSVP RESV message is received. Hence, the ability to detect such conditions and trigger the unpinning of a path is required. This can be achieved using similar mechanisms in both explicit and hop-by-hop routing cases using the approach of [GKH97]. Note that unpinning a path only implies that QoS routing is being queried anew to determine if the current path is still the correct one, or to find if a new and better one now exists. In particular, unpinning a path does not result in removal of existing path or reservation states. This is because although the existing pinned path may not fully satisfy the requirements of the flow, it may be the best one currently available. In that case any (partial) reservation that may exist on the current path should be maintained as it represents the best possible QoS available to the flow.

There are other instances where a path needs to be unpinned. For example, when one of the links or nodes on the path fails. In such cases, it is important to notify all nodes on the current path, so that they can unpin it and query QoS routing to possibly find an alternate path. This can again be achieved using similar mechanisms in both the explicit route and hop-by-hop routing cases [GKH97]. However, when a reservation is already in place, it is also desirable to identify links on which resources are already reserved for the flow. This is important so that these existing reservations be taken into account when searching for an alternate path, i.e., avoid the ``stepping on one's shadow'' problem. This is made easier in the case of explicit route by the knowledge of the entire path.

Knowledge of the entire path is also useful in the context of high level admission control, and we now briefly review this issue and the benefits of explicit routes in that context.

3.2. Enforcing high level admission control policies

As pointed out in the framework document for QoS routing [CNRS97], some form of higher level admission control and administrative control of routing behavior may be necessary within an AS. This is because QoS routing has to balance the sometimes conflicting requirements of high network resource utilization and improved chances of successful resource reservation for individual flows. For example, when current load in the network suggests a QoS path that is much longer than the ``usual'' path, admitting the flow along such a path may actually deny service to later flows that would have been admissible along segments of this long path. Hence, this could negatively affect the overall network utilization. In such situations, a high level admission control policy may find it desirable not to admit the flow based on routing decision alone. One possible approach is to compare the length of the path returned by QoS routing to that of a ``usual'' path, and decide whether or not to use the path depending on this comparison as well as possibly other factors such as overall network load. Conversely, if a flow has been already set up and later a much more efficient path becomes available, it might be desirable to reroute the flow along the new path. This is particularly true if the current path only supports a fraction of the desired reservation, while the new path may be able to accommodate the complete reservation.

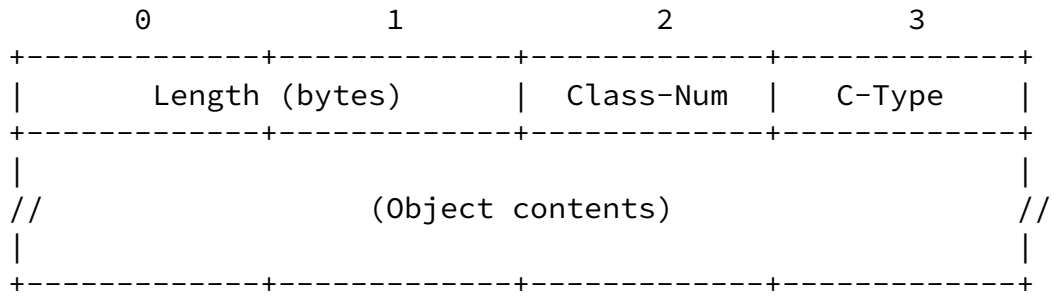
In all such instances, these decisions are greatly facilitated when a single entity is responsible for determining and controlling the entire path. Hence, such controls are more readily performed when routing is done using explicit routes instead of hop-by-hop routing. This is not to say that they are not feasible with hop-by-hop routing, but distributed decisions and knowledge generally complicate such tasks. For example, transient inconsistent routing information at multiple routers can lead to the pinning of a long but loop-free path, without any single router on the path being aware of the problem. Hence, it becomes difficult to identify and rectify such bad routing choices. Solutions to this problem require the introduction of additional signaling information to coordinate information and decisions across the nodes on the path, e.g., a policy object carried in PATH messages that specifies a limit on the acceptable path length. This would in turn add to the overall signaling and processing overhead, and may all but eliminate the potentially greater simplicity of hop-by-hop routing. On the other

hand, the single decision point of explicit routes avoids most of these problems.

4. Mechanism for Reservation Set Up on Explicit Paths

4.1. Explicit Route Object

As stated earlier, explicit routes are to be specified through a new Explicit_route object in RSVP. RSVP PATH messages will carry this object. The format of the explicit route object is described below.



Class-Num

The Class-Num indicates that the object is POLICY_DATA.

C-Type

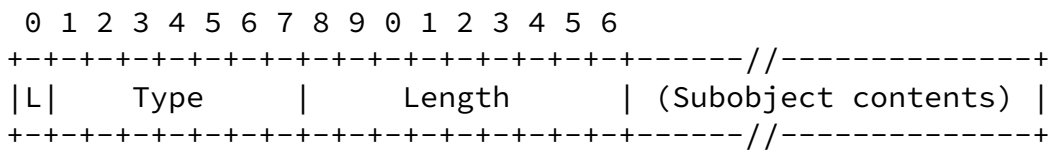
The C-Type for an Explicit Route Object is XXX [TBD].

If a PATH message contains multiple explicit route objects, only the first object is meaningful. Subsequent explicit route objects may be ignored and should not be propagated.

4.1.1. Subobjects

The contents of an explicit route object are a series of variable length data items called subobjects. Each subobject has the form:

0 1



L

The L bit is an attribute of the subobject. The L bit is set if the subobject represents a loose hop in the explicit route. If the bit is not set, the subobject represents a strict hop in the explicit route.

Type

The Type indicates the type of contents of the subobject. Currently defined values are:

0 Reserved
1 IPv4 prefix
2 IPv6 prefix
32 Autonomous system number
64 MPLS label switched path termination

Length

The Length contains the total length of the subobject in bytes, including the L, Type and Length fields. The Length must always be a multiple of 4, and at least 4.

4.1.2. Applicability

The Explicit Route Object is intended to only be used for unicast situations. Applications of explicit routing to multicast are a topic for further research.

The Explicit Route Object is only to be used when all routers along the explicit route support RSVP and the Explicit Route Object. The mechanisms for determining that such support is present are beyond the scope of this document.

4.1.3. Semantics of the Explicit Route Object

An explicit route is a particular path in the network topology. Typically, the explicit route is computed by a node, with the intent of directing traffic down that path.

An explicit route is described as a list of groups of nodes along the explicit route. Certain operations to be performed along the path can also be encoded in the explicit route.

In addition to the ability to identify specific nodes along the path, an explicit route can identify a group of nodes that must be traversed along the path. This capability allows the routing system a significant amount of local flexibility in fulfilling a request for an explicit route. In turn, this allows the generator of the explicit route to have imperfect information about the details of the path.

The explicit route is encoded as a series of subobjects contained in an explicit route object. Each subobject may identify a group of nodes in the explicit route or may be an operation to be performed along the path. An explicit route is then a path including all of the identified groups of nodes, with the specified operations occurring along the path.

To simplify the discussion, we call each group of nodes an abstract node. Thus, we can also say that an explicit route is a path including all of the abstract nodes, with the specified operations occurring along that path.

As an example, consider an explicit route that consists solely of autonomous system number subobjects. Each subobject corresponds to an autonomous system in the network topology. Each autonomous system is an abstract node. In this case, the explicit route is a path including each of the specified autonomous systems. There may be multiple hops within each autonomous system.

4.1.4. Strict and Loose subobjects

The L bit in the subobject is a one-bit attribute. If the L bit is set, then the value of the attribute is `loose.' Otherwise, the value of the attribute is `strict.' For brevity, we say that if the value of the subobject attribute is `loose' then it is a `loose subobject.' Otherwise, it's a `strict subobject.' Further, we say that the abstract node of a strict or loose subobject is a strict or a loose node, respectively. Loose and strict nodes are always interpreted relative to their prior abstract nodes.

The path between a strict node and its prior node MUST include only network nodes from the strict node and its prior abstract node.

The path between a loose node and its prior node MAY include other network nodes which are not part of the strict node or its prior abstract node.

The L bit has no meaning in operation subobjects.

4.1.5. Loops

While the explicit route object is of finite length, the existence of loose nodes implies that it is possible to construct forwarding loops during transients in the underlying routing protocol. This may be detected by the originator of the explicit route through the use of another opaque route object called the Record Route object. The Record Route object is used to collect detailed path information and is useful for loop detection as well as diagnostic purposes. The definition of Record Route object is deferred to another draft.

4.2. Subobject semantics

4.2.1. Subobject 1: The IPv4 prefix

The contents of an IPv4 prefix subobject are a 4 octet IPv4 address, 1 octet of prefix length, and 1 octet of padding. The abstract node represented by this subobject is the set of nodes which have an IP address which lies within this prefix. Note that a prefix length of 32 indicates a single IPv4 node.

The length of the IPv4 prefix subobject is 8 octets. The contents of the 1 octet of padding must be zero on transmission and must not be checked on receipt.

4.2.2. Subobject 2: The IPv6 address

TBD

4.2.3. Subobject 32: The autonomous system number

The contents of an autonomous system (AS) number subobject are a 2 octet autonomous system number. The abstract node represented by this subobject is the set of nodes belonging to the autonomous system.

The length of the AS number subobject is 4 octets.

4.2.4. Subobject 64: MPLS label switched path termination

The contents of an MPLS label switched path termination subobject are 2 octets of padding. The subobject is an operation subobject. This object is only meaningful if there is a Label Object in the PATH message.

If a Label Object is present in the PATH message, then this PATH message is being used to establish a Label Switched Path. In this case, this subobject indicates that the prior abstract node should remove one level of label from all packets following this Label Switched Path.

The length of the MPLS label termination subobject is 4 octets.

4.3. Processing of the Explicit Route Object

4.3.1. Selection of the next hop

A PATH message containing an explicit route object must determine

the next hop for this path. Selection of this next hop may involve a selection from a set of possible alternatives. The mechanism for making a selection from this set is implementation dependent and is outside of the scope of this specification. Selection of particular paths is also outside of the scope of this specification, but it is assumed that each node will make a best effort attempt to determine a loop-free path. Note that such best efforts may be overridden by local policy.

To determine the next hop for the path, a node performs the following steps:

1) The node receiving the RSVP message must first evaluate the first subobject. If the node is not part of the abstract node described by the first subobject, it has received the message in error, and should return a "Bad initial subobject" error. If the first subobject is an operation subobject, the message is in error, and the system should return a "Bad Explicit Routing Object" error. If there is no first subobject, the message is also in error and the system should return a "Bad Explicit Routing Object" error.

2) If there is no second subobject, this indicates the end of the explicit route. The explicit route object should be removed from the PATH message. This node may or may not be the end of the path. Processing continues with section 4.3.2, where a new explicit route object may be added to the PATH message.

3) Next, the node evaluates the second subobject. If the subobject is an operation subobject, the node records the subobject, deletes it from the explicit route object and continues processing with step 2, above. Note that this changes the third subobject into the second subobject in subsequent processing. The precise operations to be performed by this node must be defined by the operation subobject.

4) If node is also a part of the abstract node described by the second subobject, then the node deletes the first subobject and continues processing with step 2, above. Note that this makes the second subobject into the first subobject of the next iteration.

5) The node determines if it is topologically adjacent to the abstract node described by the second subobject. If so, the node

selects a particular next hop which is a member of the abstract node. The node then deletes the first subobject and continues processing with section 4.3.2.

6) Next, the node selects a next hop within the abstract node of the first subobject that is along the path to the abstract node of the second subobject. If no such path exists then there are two cases:

6a) If the second subobject is a strict subobject, then there is an error and the node should return a "Bad strict node" error.

6b) Otherwise, if the second subobject is a loose subobject, then the node selects any next hop that is along the path to the next abstract node. If no path exists, then there is an error, and the node should return a "Bad loose node" error.

7) Finally, the node replaces the first subobject with any subobject that denotes an abstract node containing the next hop. This is necessary so that when the explicit route is received by the next hop, it will be accepted.

4.3.2. Adding subobjects to the explicit route object

After selecting a next hop, the node may alter the explicit route in the following ways.

If, as part of executing the algorithm in section 4.3.1, the explicit route object is removed, the node may add a new explicit route object.

Otherwise, if the node is a member of the abstract node for the first subobject, then a series of subobjects may be inserted before the first subobject or may replace the first subobject. Each subobject in this series must denote an abstract node that is a subset of the current abstract node.

Alternately, if the first subobject is a loose subobject, an arbitrary series of subobjects may be inserted prior to the first subobject.

4.3.3. Error subcodes

In the processing described above, certain errors need to be reported as part of a ``Routing problem'' PathErr message. This section defines the subcodes for the errors described above.

Value	Error
1	Bad Explicit Routing Object
2	Bad strict node
3	Bad loose node
4	Bad initial subobject

5. Conclusions

This document provides a motivation for supporting opaque routing objects in RSVP to enable setting up resource reservations on explicit routes. The benefits of this approach in the contexts of MPLS and QoS routing were expounded and a mechanism for supporting this feature was discussed.

References

[CDF+97] R. Callon, P. Doolan, N. Feldman, A. Fredette, G. Swallow, and A. Viswanathan. A Framework for Multi-Protocol Label Switching (draft-ietf-mpls-framework-00.txt). INTERNET-DRAFT, Internet Engineering Task Force, May 1997.

[CNRS97] E. Crawley, R. Nair, B. Rajagopalan, and H. Sandick. A Framework for QoS-based Routing in the Internet (draft-ietf-qosr-framework-00.txt). INTERNET-DRAFT, Internet Engineering Task Force, March 1997.

[GKH97] R. Guerin, S. Kamat, and S. Herzog. QoS Path Management with RSVP, (draft-guerin-qos-path-mgt-rsvp-00.txt). INTERNET-DRAFT, Internet Engineering Task Force, March 1997.

[GKO+97] R. Guerin, S. Kamat, A. Orda, T. Przygienda, and D. Williams. QoS Routing Mechanisms and OSPF Extensions, (draft-guerin-qos-routing-ospf-01.txt). INTERNET-DRAFT, Internet Engineering Task Force, March 1997.

[GKR97] R. Guerin, S. Kamat, and E. Rosen. Extended RSVP-Routing Interface (draft-guerin-ext-rsvp-rtng-intf-00.txt). INTERNET-DRAFT, Internet Engineering Task Force, July 1997.

Authors' Address

Der-Hwa Gan
Juniper Networks, Inc.
385 Ravendale Dr.
Mountain View, CA 94043
Email: dhg@juniper.net
Phone: +1 650 526 8074
Fax: +1 650 526 8001

Roch Guerin
IBM T.J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598
EMAIL: guerin@watson.ibm.com
VOICE +1 914 784-7038
FAX +1 914 784-6205

Sanjay Kamat
IBM T.J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598
EMAIL: sanjay@watson.ibm.com
VOICE +1 914 784-7402
FAX +1 914 784-6205

Tony Li
Juniper Networks, Inc.
385 Ravendale Dr.
Mountain View, CA 94043
Email: tli@juniper.net
Phone: +1 650 526 8006
Fax: +1 650 526 8001

Eric Rosen
Cisco Systems, Inc.
250 Apollo Drive
Chelmsford, MA, 01824
EMAIL: erosen@cisco.com