Instructions for part3: Interfaces and next-hops

When a router forwards packets, it sends them out of a "local" interface to a "next-hop" interface that is closer to the destination. In a point-to-point link (for example an ATM link) the next-hop will be the interface at the other end of the link. In a broadcast link (for example Ethernet) the next-hop interface will be one of the interfaces that are on the same subnet as the local interface. When a router needs to find where to send a packet it consults the routing table where it finds both the local and the next-hop interfaces that have to be used for a particular destination. The concepts of the local and next-hop interfaces are also used in the protocols. In this part of the project we will see how RSVP-TE deals with next-hop interfaces.

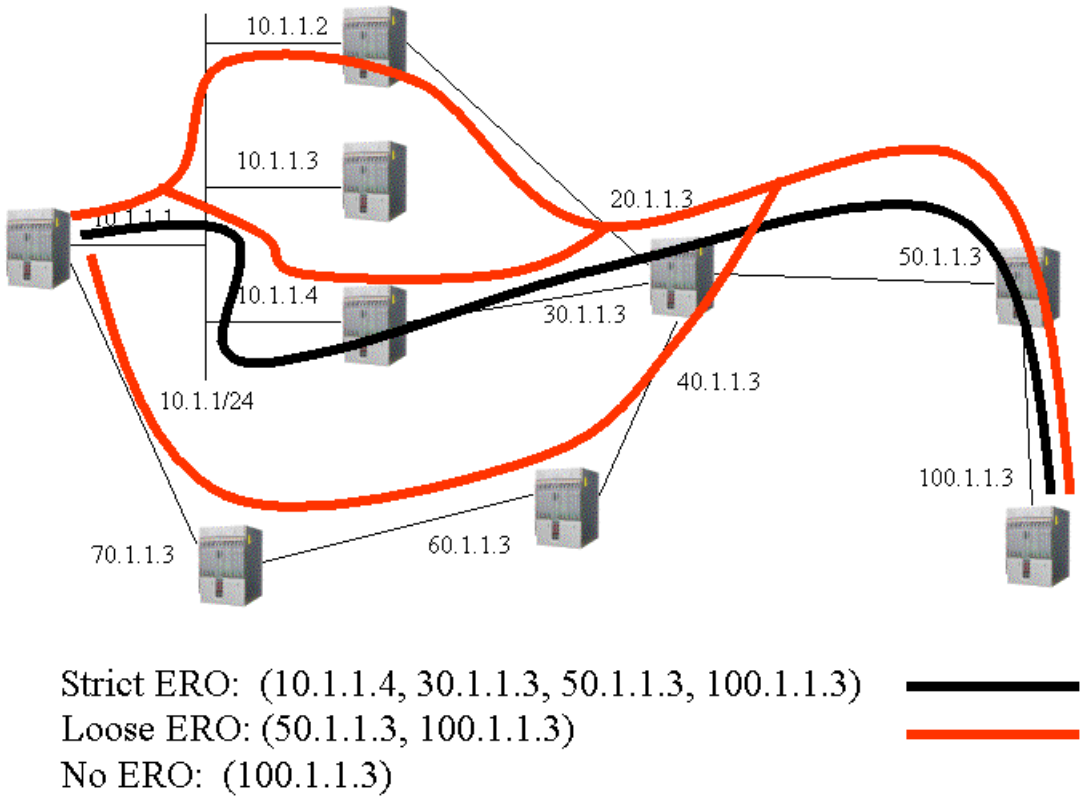

Figure 1

Remember we have three ways to tell RSVP where to setup the LSP.

1. Without an explicit route. In this case we specify only the destination IP address of the egress of the LSP. The PATH messages will have this address as the destination address in the packet. The Router Alert option will be set. Each RSVP node will forward the PATH packet towards the destination using the information available through the IGP routing.
2. With a strict explicit route. In this case, all the intermediate nodes in the path towards the destination are specified in the explicit route. Each node in the path knows which is the IP address of the interface of the next node in the path and sends the PATH message there.

There is no need to set the router alert option in the packet. If the next hop is down the LSP can not be setup.

3. With a loose explicit route. A loose explicit route specifies a subset of the intermediate nodes in the path but not all of them. Thus some nodes will know exactly which is the next hop (like in the strict explicit route case) and some nodes will not know their next hop and will have to find this information from the IGP.

In RSVP it is possible to have an explicit route that have some strict and some loose hops. For example in Figure 1, the "loose" explicit route, has a loose next-hop (50.1.1.3) and a strict next-hop (100.1.1.3). There is a bit in the explicit route that specifies if the particular hop is strict or loose.

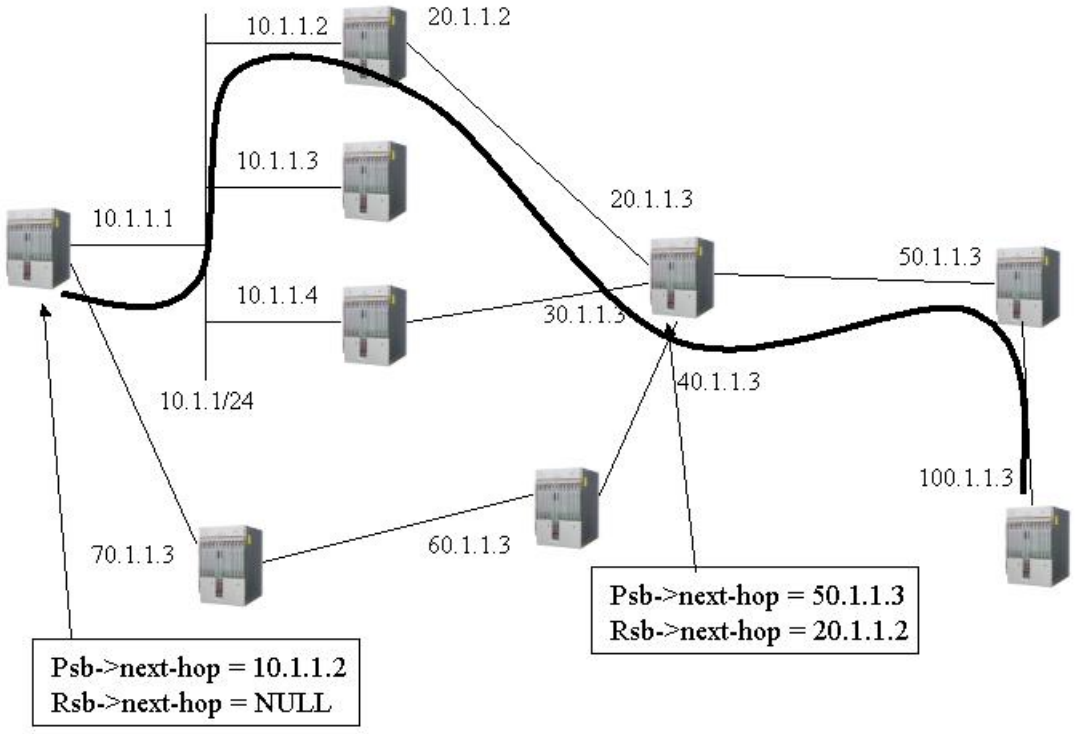

Figure 2

In this part of the project, we will work with loose next-hops and how to retrieve information about them from the IGP. In order to do that, RSVP must be able to find out what are the contents of the routing table so it can determine the local and next-hop interface to use for reaching the destination address. In Quagga, this requires rsvpte to talk to zebra since the zebra process knows all the routes in the system. The zebra protocol provides a special type of message for this: ZEBRA_IPV4_NEXTHOP_LOOKUP. A zebra client (rsvpte in our case) sends zebra the destination address and zebra will reply with a struct nexthop (defined in file zebra/rib.h). The next-hop information returned by zebra can be either the IP address of the local interface or its ifindex. You can find an example how this is used in bgp/bgp_nexthop.c. BGP also needs to find

out what interface to use for reaching remote addresses, remember that when a BGP router receives a path over an iBGP connection, the BGP next-hop for the path is the address of the iBGP peer router, who is multiple hops away. BGP needs to know how to reach this remote BGP router. RSVP stores the next-hop information in the rsb and psb structures. For each LSP/session there will be one downstream next-hop (for sending PATH messages to) stored in the psb structure and one upstream next-hop (used for sending the RESV messages to) stored in the rsb structure. These next-hops are determined differently though. The downstream next-hop is determined by the local router using the explicit route information. If it is a strict next-hop, we use the value contained in the explicit route. If it is a loose next-hop, we have to ask the IGP. The upstream next-hop is where we received the PATH message from. Each incoming PATH message contains a hop object that tells me the IP address of the interface that sent the message. Thus, we will need to query the IGP only for the downstream loose next-hops.

Having to find out IGP route information from RSVP (and BGP) causes some problems. First, it makes RSVP and BGP dependent on the IGP. If the IGP has not converged, RSVP can not work properly. Then, when IGP changes paths (if a link fails or the cost of links changes) RSVP needs to be notified so that it changes the next-hops it is using. Since we do not know when the IGP next-hops will change, RSVP will need to resolve the next-hops periodically. This is the **pull** model which results in some latency until an IGP change is detected. Other systems implement a **push** model. Their RIB process (the equivalent of zebra) implements a notification service. In these systems RSVP would register a destination with RIB and RIB would notify it when the next-hop to this destination changes. This may be more scaleable in cases where there are a lot of next-hops. In a large network there can be 100s of next-hops and resolving them every few seconds is a lot of unnecessary work.

Interface up/down

All protocols need to monitor the state of the interfaces and take appropriate action when they fail. In RSVP the action depends on the type of the next-hop that uses this interface. If we have a strict next-hop (i.e. there is a strict explicit route that requires us to use this interface) we can not do much. The LSP can not continue to function and we have to send a RESVerr upstream. If the next-hop is a loose one, we have to resolve it again and start using a different interface. The LSP is still up but it follows a different path now and we should start sending PATH messages down the new path. Of course, when the interface fails, we will have to walk all the sessions and find the ones that use this interface for their next-hops and update them. It is possible that the upstream interface will fail. In this case, we can not repair the LSP and we will need to send PATHerr messages downstream.

Organization of the data structures

In Figure 3 we show how the rsvp data structures should be organized. Sessions are organized in a tree. Each session has exactly one psb and one rsb which is unique to the session (no two sessions can have the same psb/rsb). The psb and rsb contain pointers to the rsvp_nhop_t data structure but many psbs/rsb may share next-hops (in general there will be relatively few next-hops in the network compared to the potentially thousands of sessions). Next-hops are organized into another tree.

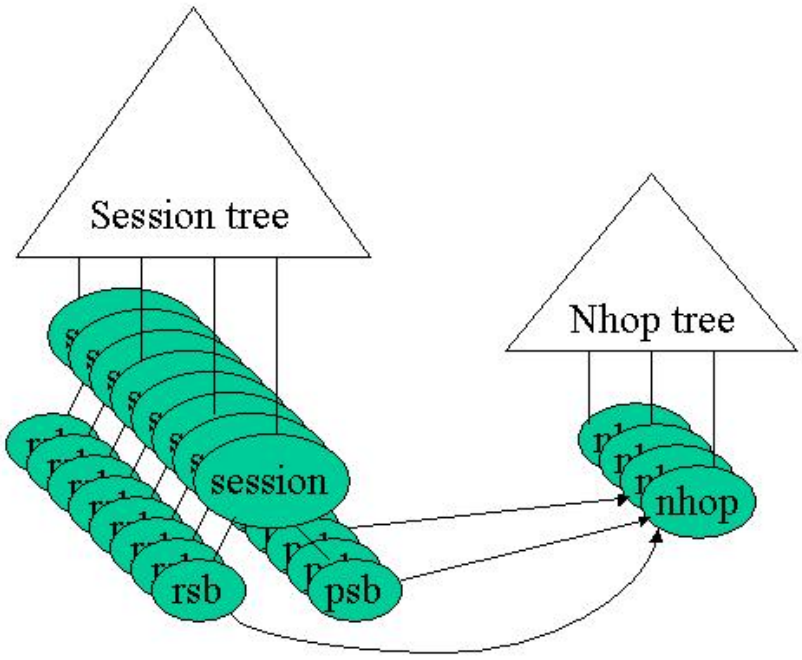

Figure 3

What to do in this phase:

- Implement the resolution of next-hops using the pull model supported by zebra. The resolution will be done using the ZEBRA_IPV4_NEXTHOP_LOOKUP zebra message similarly to how BGP uses it. Since Quagga implements only the pull model, you will need to put all the next-hops in a tree and resolve them periodically. In order to ensure that we are scalable we should plan for the case where we have too many next-hops. Thus we will need to use a walk for the next-hops that will do a limited amount of work and will stop in order to allow other protocol work. Each time the interface changes, we should originate a PATH message over the new interface. It may even be possible that the next-hop is not reachable anymore (if link failures caused the network to partition and there are no IGP routes for the next-hop). In this case, we should send an error upstream and bring the LSP down. The next-hops are stored in ps_rsvp_nhop in struct rsvp_psb_t for the next-hop, and ps_rsvp_phop in struct rsvp_psb_t for the previous hop. For now ignore struct rsvp_rsb_t. These nhops are of type struct rsvp_hop_t that is a rsvp HOP object (i.e. includes the object header). Struct rsvp_hop_t is defined in file rsvp_packet.h.

- Handle interface down/up events. When an interface changes state, function `rsvp_interface_state_down()` (and `rsvp_interface_state_up()` for interface up

events) in file rsvp_zebra.c will be called. Then we need to walk the session tree, find with sessions use this interface for downstream or upstream traffic, and act accordingly. If the next-hop is loose then we will need to re-resolve the next-hop interface and local interface from zebra. If the next-hop was not loose then we will need to send error messages. Here for simplicity we will assume that all the next-hops are loose, so we will always resolve them with zebra. If the resolution does not return a valid local interface and next-hop then we will have to send error messages. When an interface becomes active again, in principle we will have to re-resolve all the next-hops in all sessions to see if there is a better local/next-hop interface for them.

Details

- Make sure you get the latest code from CVS (as of Friday 1st of December). It has changes to set the Router Alert option correctly.
- We assume that all downstream next-hops are loose. All the PATH messages sent have the Router Alert option set.
- Do not worry about sending PATHerr and RESVerr messages they are not yet implemented in the code.
- You will need to modify function find_outgoing_interface() defined in rsvp_zebra.c to look at the destination information for the LSP and figure out the local interface and the next-hop interface to use.
- Assume that we do not have explicit routes (loose or strict). The only information that you will need to look at in function find_outgoing_interface() is the destination address of the LSP. You can find this from the session for the LSP
- You will need some IGP routing in order to try this part. We will use OSPF for IGP routing. For en example how to configure OSPF look at the machines godzilla13, 14 and 15 (behind ports 2205, 2206, 2207) they are configured to run OSPF and form a small network, with eth1 of godzilla13 connected to eth1 of 14 and eth2 of 14 connected to eth2 of 15. The configuration files for OSPF can be found in /usr/local/etc/ospfd.conf in each machine. You can connect to the OSPF process in each machine with telnet localhost 2604, password “zebra”. There you can do “show ip ospf neighbors” to see the adjacencies with the other routers. **Attention**: by default Linux systems do not perform packet forwarding and will not work as routers. You need to do “sysctl net.ipv4.ip_forward=1” in a machine to make it work as router.