Integrated Services Provisioning Across Cryptographic Boundaries

Orlie T. Brewer, Arun Ayyagari, and Michael S. Foster

Boeing Phantom Works
P.O. Box 3707, MC 45-18, Seattle, WA 98124-2207
206-655-5996 (TEL), 206-655-4874 (FAX)
{orlie.t.brewer; arun.ayyagari; michael.s.foster}@boeing.com

ABSTRACT

IntServ Resource ReSerVation Protocol (RSVP) is based on end-to-end signaling and the current HAIPE specification does not allow for RSVP signaling to be bypassed across cryptographic boundaries. Since end-to-end RVSP signaling traffic is not bypassed across HAIPE boundaries, it does not seamlessly allow for IntServ based QoS provisioning within the core Black network. This leads us to the challenge of defining a mechanism by which IntServ/RSVP can be supported within the core Black network. We built upon our prior work on a dynamic DiffServ network QoS management framework developing an IntServ implementation that operates across HAIPE boundary. The objective of our effort was to allow for individual IntServ/RSVP sessions in the red security enclave to be aggregated into a finite set of dynamically instantiated IntServ/RSVP sessions between ingress and egress nodes within the black security enclave. We used simple policy based management whereby the RSVP daemon on the ingress black node monitors the DSCP values on its outbound ports to initiate the creation or deletion of aggregated IntServ/RSVP sessions to the appropriate egress black node. These egress black node sessions are dynamically resized based on traffic demand and network state. This approach allowed for end-to-end IntServ across HAIPE boundaries.

Keywords: QoS, DiffServ, IntServ, RSVP, IP Security, HAIPE, dynamic ad hoc mobile heterogeneous networks

I. INTRODUCTION

Department of Defense (DoD) policies require the use of High Assurance Internet Protocol Encryptor (HAIPE) devices that provide cryptographic isolation between data in red security enclaves and data that is transported across a black shared transit network through HAIPE tunnels, i.e., IP Security (IPSec) tunnel mode with Encapsulating Security Payload (ESP). As a result, packet exchanges and even IP addresses visible in the red enclaves are opaque to the black network. This segmentation of the network at cryptographic boundaries impacts the operation of QoS mechanisms, most of which require signaling messages to be passed between peer network elements. The QoS mechanism that is presently compatible with HAIPE is Differentiated Services (DiffServ). While DiffServ provides Per Hop Behavior (PHB) management, end-to-end QoS provisioning via Integrated Services (IntServ) across both the red and black security enclaves is required for specific real-time traffic flows. IntServ enables per domain behavior (PDB) provisioning for a session via end-to-end QoS provisioning from the source to the destination nodes. A key challenge in the deployment of IntServ within current HAIPE specification environment is that only the Type of Service (ToS) byte in IPv4, which includes the 6-bit DiffServ Code Point (DSCP) and the 2-bit Explicit Congestion Notification (ECN) may be bypassed across cryptographic boundaries while IntServ Resource ReSerVation Protocol (RSVP) signaling cannot be bypassed. Since RVSP signaling traffic is not bypassed, it does not seamlessly allow for IntServ based QoS provisioning within the core Black network. We built upon our prior work on a dynamic DiffServ network QoS management framework by developing an IntServ implementation that operates across HAIPE boundary [1]. The objective of our effort was to allow for individual IntServ/RSVP sessions in the red security enclave to be aggregated into a finite set of dynamically instantiated IntServ/RSVP sessions between the ingress and egress nodes within the black security enclave.

II. INTSERV PROVISIONING OVER DIFFSERV NETWORKS

IntServ enables PDB provisioning for a session via end-to-end QoS provisioning from the source to the destination nodes [2]. IntServ uses a protocol called RSVP by which applications can request end-to-end per-conversation QoS from the network, and can indicate QoS requirements and capabilities to peer applications [3]. As illustrated in Figure 1, we have integrated RSVP-based IntServ implementation (IETF RFC 2210) into our existing QoS Service Provider framework. The QoS request signaled via RSVP is mapped to a specific filter and an appropriate policer within the underlying DiffServ Traffic Control (TC) scheduler [4].

In order to incorporate the use of IntServ into our QoS Service Provider, we used the KOM RSVP environment developed at the Technical University of Darmstadt in Darmstadt, Germany [5]. This environment consists of an RSVP protocol engine implemented as a user space daemon with a client API. The code was written in C++ and has been compiled and tested on Linux, Solaris, and FreeBSD. For a description of using RSVP with Integrated Services, see IETF RFC 2210.

There are two situations when the QoS Service Provider and the RSVP daemon must interact. The first is when the QoS Service Provider is requested to use IntServ for a network connection. The second is upon the receipt of a RSVP RESV message by the RSVP daemon. The first point of interaction required functions

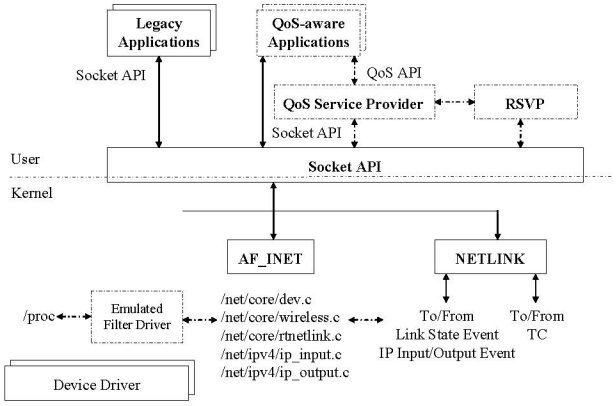

Figure 1: Network QoS Management Architecture

to allow the QoS Service Provider, written in C, to be able to use the C++ API of the RSVP daemon. We accomplished this by using the extern "C" function qualifier in C++, to define functions that could be called from the C code of the QoS Service Provider, but could be compiled in C++ and could thus call the C++ functions of the RSVP API. The second point of interaction was written entirely in C++ within the RSVP daemon. These two situations will be described in detail below.

An application using the QOSAPI may request that IntServ be used for the network connection by setting the qosmech field, in the qos_info data structure, to INTSERV or alternatively the QoS levels requested for the network connection may be mapped to INTSERV by the QoS Service Provider. The QoS Service Provider must first be configured to support IntServ by specifying which DiffServ class will be used for IntServ traffic. This is done by placing a statement of the form "RSVP_CLASS AF4" in the DiffServ configuration file used to invoke the QoS Service Provider. This class will be reserved for IntServ traffic. This statement should be consistent across all hosts/routers using RSVP.

Upon receiving the request from the sending application, the QoS Service Provider will create a filter to map that flow into the DiffServ class as defined above. It will then use the API of the RSVP daemon to create a sender object. This action will initiate the sending of a RSVP PATH message to the destination. At the destination, there must be a receiving application running that has registered with the RSVP daemon on that host that it will accept RSVP PATH messages to that IP address, port number and protocol. This receiving application need not be the same process as the one that will actually accept the network connection. Thus, legacy applications can be used with only a small separate proxy application to handle the RSVP signaling. It would be an easy task to design a proxy application that could handle the RSVP signaling for any number of legacy applications by reading appropriate information from a configuration file.

The job of the receiving application is to take the RSVP PATH message and, using the API of the RSVP daemon on that host, request the creation of a reservation. This action will initiate the sending of a RSVP RESV message hop-by-hop back to the sender along the reverse path of the PATH message. At each hop, upon receipt of the RESV message, the RSVP daemon will check the scheduler on that interface for admission control. This is the second point of interaction between the QoS Service Provider and the RSVP daemon.

The scheduler interface in KOM RSVP consists of an abstract class called BaseScheduler with five pure virtual functions that must be defined for a concrete class. These five functions initialize a scheduler, add a flowspec, delete a flowspec, add a filter, and delete a filter. Specific classes such as CBQ and HFSC were subclassed from BaseScheduler. Using these subclasses as examples, we subclassed the SchedulerQOSSP class from the BaseScheduler class. An object of this class will communicate to the QoS Service Provider over a UNIX socket for admission control and filter creation. The QoS Service Provider keeps track of how much bandwidth has been allocated for IntServ flows and also knows the bandwidth that was initially allocated to the DiffServ class reserved for IntServ. It uses this information for admission control decisions. The filter for this flow is placed at the top level of the DSMARK QDISC for the DiffServ class used for IntServ, see Figure 2. The filter has its own policer for this flow. As long as the flow stays within its requested bandwidth, that flow will be mapped to the lowest dropping level for that DiffServ class. However, if that flows exceeds its requested bandwidth, any excess traffic will be mapped to a higher dropping precedence level in that class. The SchedulerQOSSP implements only the controlled-load service, see IETF RFC RFC2211 [6].

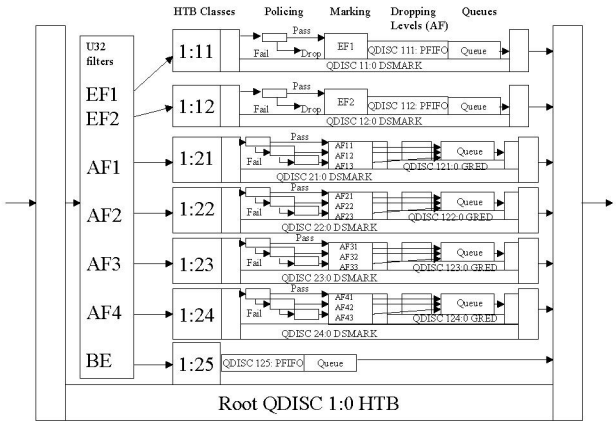

Figure 2: DiffServ implementation using Linux TC elements

If the RESV message succeeds at every hop on the way back to the sending host and if the application has specified that it wants to be notified of events of the type RESV_RECEIVED_EVENT, the RESV message will be sent to a callback routine defined by the application. If admission control fails at any point along the path back to the sender, the RSVP daemon on that host stops transmission of the RESV message at that point and sends a RESV_ERROR message back to the receiving proxy. The receiving proxy will release the original reservation, causing a

RESV_TEAR to be sent hop-by-hop back to the host generating the RESV_ERROR. The RSVP daemon will also notify the QoS Service Provider on that host to send a message to the QoS Service Provider on the host originating the RSVP PATH request, notifying it of the RESV_ERROR. The QoS Service Provider on the originating host will notify the application's callback of the event if the application has requested to be notified of events of the type RESV_ERROR_EVENT. If the QoS Service Provider on the originating host has not received a RESV message or a RESV_ERROR message within a configurable amount of time, it will call the application's callback if the application has specified that it wants to be notified of events of the type RESV_NOT_RECEIVED_EVENT.

Upon the application calling QClose for the socket, the QoS Service Provider will use the RSVP API to release the sender object originally created on opening the connection. This action will initiate the sending of a PATH_TEAR message to the destination. At each hop along the path to the destination, the RSVP daemon will release all resources for that flow and inform the scheduler to remove all references to that flow. The SchedulerQOSSP will send this information to the QoS Service Provider, which will remove the filter for that flow.

If the QoS Service Provider receives notification of a link change, while it has been configured for supporting IntServ, it will first adjust the rates allocated to the different Differentiated Services classes. It will then compare the newly calculated rate of the Differentiated Services class reserved for RSVP with the amount of bandwidth currently allocated to RSVP flows. If the new rate is still greater than or equal to what is currently allocated, the QoS Service Provider will do nothing. If, however, the new rate is less than the amount currently allocated to RSVP flows, it will make some adjustments.

It will calculate how much to reduce the rate for each RSVP flow proportionately in order for the sum of the flows to be equal to the new rate of the Differentiated Services class reserved for RSVP. Then it will adjust the rates of the policers attached to the filters for each RSVP flow. In case if a fluctuating bandwidth, it will wait a configurable amount of time to see if the bandwidth of the link increases to a level sufficient for all RSVP flows to have their originally requested rates, in which case it will adjust the rates on the policers to their original rates. However, if the rate remains too low, it will notify the QoS Service Provider on the host of the sender of each RSVP flow that its rate on this node has been reduced, using the TCP socket reserved for communications between QoS Service Providers.

III. INTSERV PROVISIONING ACROSS HAIPE BOUNDARIES

DoD policies require the use of HAIPE devices that provide cryptographic isolation between data in red security enclaves and data that is transported across a black shared transit network through HAIPE tunnels, i.e., IPSec tunnel mode with ESP, see Figure 3. As a result, packet exchanges and even IP addresses visible in the Red enclaves are opaque to the Black network. This segmentation of the network at cryptographic boundaries impacts the operation of QoS mechanisms, most of which require signaling messages to be passed between peer network elements. The QoS mechanism that is presently compatible with HAIPE is DiffServ. While DiffServ provides PHB management, end-to-end QoS provisioning via IntServ across both the red and black security enclaves is required for specific real-time traffic flows.

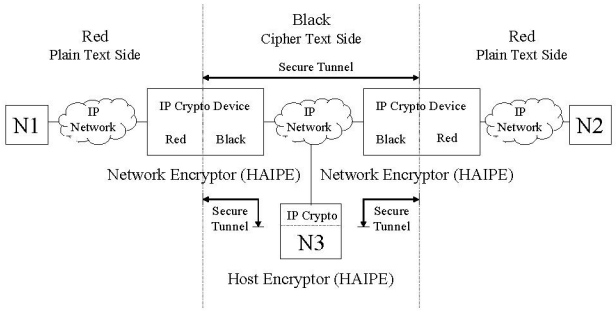

Figure 3: Information Security Deployment Topology

IntServ enables PDB provisioning for a session via end-to-end QoS provisioning from the source to the destination nodes. Our current capability to dynamically create and remove DiffServ policer enables us to map the IntServ session to a unique policer or can be mapped to an existing DiffServ class policer if an aggregated flow control is acceptable. This development provides the framework for further extension to incorporate aggregation of RSVP sessions (similar to IETF RFC 3175) at specific intermediary nodes within the infrastructure [7,8,9]. Within JTRS, FCS, TCA networks the ideal place for these aggregation and deaggregation points are the ingress and egress Red/Black boundaries within High Assurance Internet Protocol Interoperability Specification (HAIPIS) environments. Scalability of RSVP-based IntServ is an issue in particular when traversing across Red/Black boundaries within a HAIPIS environment. Aggregation of RSVP reservations provides a means of aggregating individual RSVP reservations into a single RSVP reservation across a transit routing region. This routing region is akin to virtual paths and this approach readily applies to the HAIPE environment across the ingress and egress Red/Black boundaries.

A key challenge in the deployment of IntServ within a HAIPE environment is that currently (HAIPIS Version 2.0) only the ToS byte in IPv4, consisting of 6-bit DSCP and 2-bit ECN, may be bypassed across the Red/Black boundaries. RSVP is based on end-to-end signaling and the current HAIPE specification does not allow for RSVP signaling to be bypassed across the Red/Black and Black/Red boundaries. The rational for this restriction is to minimize the amount of plain-text information that can be bypassed across the Red/Black and Black/Red security enclaves to maintain desired levels of Information Assurance against security threats such as traffic analysis and Denial of Service (DoS) attacks. Since RSVP signaling traffic is not bypassed it is carried within HAIPE tunnels it does not allow for QoS provisioning within the core Black network. This leads

to the challenge of defining a mechanism by which RSVP-based IntServ can be supported within the core Black network.

The objective is to provision RSVP-based IntServ within the HAIPE based core Black network as a function of the RSVP-based IntServ sessions between Red network clients separated by a core Black network. An approach to accomplishing this via explicit signaling is by allowing for in-band bypass of RSVP signaling across Red/Black and Black/Red HAIPE boundaries. Alternatively one may support out-of-band secure private/proprietary control signaling across the Red/Black HAIPE boundary. In both cases there is a need for explicit transfer of control information across the HAIPE boundary that is currently not permitted. While the long-term goal is to achieve greater resource management control, it would require approval from government National Security Agency (NSA). NSA would likely pose barriers due to the additional exposure of plain-text information within the core Black network. As an alternative, we have defined an implicit signaling mechanism by which RSVP-based IntServ can be supported in the core Black network, within current HAIPE specification constraints for the bypass of information across the Red/Black and Black/Red boundaries. Our approach leverages the bypass allowed in the current HAIPE specification, 6-bit DSCP and 2-bit ECN across the Red/Black boundary and 2-bit ECN across the Black/Red boundary. This approach does not require any additional control signaling bypass while allowing for provisioning of RSVP-based IntServ within the HAIPE based core Black network as a function of the RSVP-based IntServ sessions between Red network clients separated by a core Black network. This design and software implementation will enable end-to-end QoS provisioning, via RSVP-based IntServ, for real-time applications within JTRS, FCS, TCA networks.

We build upon the dynamic DiffServ network QoS management framework by developing IntServ implementation that operates across HAIPE boundary. The goal of this task is to allow for individual RSVP-based IntServ sessions on the red security enclave to be aggregated into a finite set of dynamically instantiated RSVP-based IntServ sessions between the ingress and egress nodes within the black security enclave. We propose to use policy based management whereby the RSVP daemon on the ingress black node would monitor the DSCP values on its outbound ports to initiate the creation or deletion of aggregated RSVP-based IntServ sessions to the appropriate egress black node. This proposed approach allows for end-to-end IntServ across HAIPE boundaries. The DSCP values that would trigger RSVP signaling within the black security enclave would be based on the policy established during mission planning to ensure consistent classification and policing within red and black security enclaves. Following the initial development, we will extend the end-to-end QoS provisioning functionality by leveraging the ECN bypass across red/black boundaries and traffic monitoring on the egress/ingress red boundaries to dynamically resize the aggregated RSVP-based IntServ session's resource allocations within the black security enclave.

The purpose of this Network QoS Management development effort is to support aggregation of red-side RSVP traffic across black networks in HAIPIS environment. In order to develop code for traffic traversing across Red/Black boundaries, we have simulated HAIPE encryption with IPSec (tunnel mode with ESP) on a stand-alone Linux machine running the 2.6 kernel (standard Fedora Core 2). We used the inline IPSec implementation of the Linux 2.6 kernel, because FreeS/Wan under the Linux 2.4 kernel did not support IP packet options, which the RSVP messages require. The inline IPSec of the Linux 2.6 kernel supports IP packet options. This simulated HAIPE device does not run any of the code for the QoS Service Provider or RSVP daemon; it merely simulates the HAIPE encryption functionality. Each simulated HAIPE device has two interfaces, one red and one black. The IPSec is configured so the red network is hidden from the black side. All red traffic on the black network is ESP encrypted and has source and destination IP addresses of the black interfaces of the simulated HAIPE devices.

Figure 4 illustrates our testbed consisting of seven machines. All simulated HAIPE devices are running the Linux 2.6 kernel (standard Fedora Core 2) and all others are running the Linux 2.4 kernel (standard Fedora Core 1). We have the interface of a red client connected to the same subnet as the red interface of a simulated HAIPE device. The black interface of the simulated HAIPE device is connected to one interface in the first black router. The second interface of this router is connected to an intermediate black router, which is also connected the last black router. This last black router is connected to the black interface of the second HAIPE device. The red interface of this simulated HAIPE device is connected to a second red client.

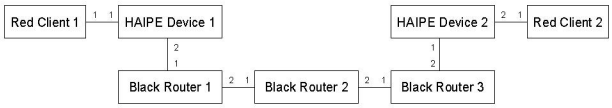

Figure 4: Network QoS Management Testbed

We want the QoS Service Provider to have the ability to detect RSVP traffic coming from the red network onto the black network. So, the first black router between the HAIPE encryption and the black network needs to be able to detect RSVP traffic. Since we are using a specific AF class for RSVP traffic and the DSCP value is copied through the IPSec encryption, the kernel needs to be able to detect packets with a source IP address of the simulated HAIPE device being routed to an interface connected to the black network with DSCP values of any of the three dropping levels corresponding to the AF class reserved for RSVP traffic.

When the QoS Service Provider has been informed by the kernel that it has seen packets with the appropriate DSCP values, we want it to set up a black-side RSVP session between the first black router and the last black router. This requires a table of black-router-haipe-device connections, so that the QoS Service Provider knows which black router to use as an RSVP endpoint when it has a simulated HAIPE device destination address. Also,

all intermediate black routers need this information when setting up the black-side RSVP session, because, in order to set up the TC filters, it will need to know the simulated HAIPE device IP addresses, since they are what will be in the IP packet headers. If the simulated or real HAIPE encryption is such that the source IP address is the first black router, then this table would be unnecessary. The table is a compile-time option in the QoS Service Provider.

Also, we want the QoS Service Provider to tear down this black-side RSVP session when there are no packets flowing through that filter for a certain amount of time.

Kernel modifications

In order for the kernel to detect and report packets with a certain source IP address and DSCP value, modifications to the kernel were necessary. We extended the rtnetlink mechanism by adding the message types RTM_NEWQOSSP, RTM_DELQOSSP, and RTM_GETQOSSP, by adding a broadcast group RTMGRP_QOSSP_DSCPWATCH and by defining the qosspmsg structure in /usr/src/linux/include/linux/rtnetlink.h. We added the function to process these message types in /usr/src/linux/net/core/rtnetlink.c. When an application sends a RTM_NEWQOSSP message to the kernel, for a particular interface, with a DSCP value and a source IP address, this function will store these values, currently in a static array of 10. Next, we added code to /usr/src/linux/net/ipv4/ip_forward.c that will search this array whenever a packet is being forwarded. If there is a match, the code saves the destination address, currently in a static array of 10, so that it will not report this match again. It then calls the function rtmsg_qosspinfo, in /usr/src/linux/include/linux/rtnetlink.h which reports the interface, DSCP value, source IP address and destination IP address to the broadcast group RTMGRP_QOSSP_DSCPWATCH.

An application can reset the monitoring of that DSCP value from the source IP address to the destination IP address by sending a RTM_NEWQOSSP message to the kernel with those values. Sending a RTM_NEWQOSSP message with just a DSCP value and source IP address, resets the monitoring of that combination to all destination addresses. Sending a RTM_DELQOSSP message to the kernel with a DSCP value and source IP address, removes that combination from the array.

Functions of the QoS Service Provider

The functions that the QoS Service Provider performs in order for all this to function, depends on where it is in the environment.

The QoS Service Provider on any red client does not need to do anything special other than ensure that the DSCP value is set to the appropriate AF class reserved for RSVP. In order to tell the QoS Service Provider which DiffServ AF class to use for RSVP traffic, the RSVP_CLASS directive in the QoS Service Provider's configuration file is used, as in RSVP_CLASS AF4. This should be the same for all interfaces throughout the environment on both the red side and the black side.

All black routers need the black-router-to-haipe-device table. This is defined in the QoS Service Provider's configuration file by using the BRHD directive as:

> BRHD <IP address of Black Router 1> <IP address of HAIPE Device 1>
> BRHD <IP address of Black Router 3> <IP address of HAIPE Device 2>

These entries should all be the same for all black routers in the environment. After the QoS Service Provider has completely read the configuration file, it will retrieve the IP address of all its interfaces and, if any of them are in the black-router-to-haipe-device table, it will mark them as local.

For black routers connected to simulated HAIPE devices and the black network, we need to tell the QoS Service Provider which interfaces are connected to the black network. The WATCH_FOR_RSVP_DSCP directive in the QoS Service Provider's configuration file is used to tell the QoS Service Provider that an interface is connected to the black network and that it needs to look for encrypted RSVP traffic coming from other interfaces on the system that are connected to simulated HAIPE devices.

If the QoS Service Provider sees the directive WATCH_FOR_RSVP_DSCP for an interface when reading its configuration file, it will mark that interface as one to watch. After completely reading the configuration file, for each interface so marked, the QoS Service Provider will look through the black-router-to-haipe-device table. For all local connections in the table, it will set the kernel to watch for packets being routed to that interface from the IP address of the simulated HAIPE device with DSCP values of the three dropping levels of the AF class reserved for RSVP.

The QoS Service Provider will then set up a listening netlink socket for the RTMGRP_QOS_DSCPWATCH broadcast group.

In order for the endpoint black router to accept an RSVP PATH message from another black router and generate an RSVP RESV message along the return path, the QoS Service Provider will setup a listening API on port 9999 with the RSVP daemon, on any interface connected to a simulated HAIPE device.

Whenever, the kernel notifies the QoS Service Provider that it has seen a packet with a certain DSCP, source IP address, and destination IP address, the QoS Service Provider will check the black-router-to-haipe-device table and attempt to setup an RSVP session between port 9998 on itself and port 9999 on the black router in the table corresponding to the destination IP address. When the RSVP PATH message reaches the last black router, it is forwarded to the QoS Service Provider, which initiates the generation of an RSVP RESV message. At each black router along the return path, if the session is accepted and if the ports

are 9998 and 9999, the QoS Service Provider will consult the black-router-to-haipe-device table to extract the simulated HAIPE device IP addresses to create a TC filter. When the original black router receives the RSVP RESV message and has created the TC filter, it will begin to monitor the filter. Because the RSVP daemon on the red-side will send REFREASH messages every twenty to thirty seconds, if there is no traffic through the filter for 100 consecutive seconds, the QoS Service Provider will assume that the red-side RSVP session has been closed and will generate an RSVP PATH TEAR message for the black-side RSVP session.

The QoS Service Provider can dynamically adjust the bandwidth reserved for an aggregated RSVP flow through the black network based on the amount of RSVP traffic coming from the red network for that particular flow. The QoS Service Provider defines an aggregated RSVP increment as a percentage of the rate assigned to the differentiated services class reserved for RSVP through the black network. This percentage is ten percent by default, but can be set to a different value in the QoS Service Provider's configuration file. The aggregated RSVP session is initially created with a flow rate of that percentage. The QoS Service Provider will adjust the bandwidth reserved for the aggregated RSVP flow up or down by that increment.

Every five seconds, the QoS Service Provider gathers statistics. It keeps a list of aggregated RSVP flows and will check the policiers for those flows. If there are any overlimit packets in an aggregated flow, the QoS Service Provider will request the RSVPD to increase the bandwidth for that flow by one increment. If the aggregated flow rate is below the current assigned rate minus the increment, it will request to decrease the bandwidth for that aggregated flow by one increment. If the aggregated flow is zero, it will request to decrease the bandwidth for that flow to the lowest level, if it is not already there. If the aggregated flow remains at zero for a configurable amount of time, one hundred seconds by default, it will remove the RSVP session.

The QoS Service Provider always saves the information for the current reservation so that if any resize request should fail, it will revert back to the last successful reservation. Specifically, when the QoS Service Provider on a black router receives the first PATH message from another black router for an aggregated RSVP session, it will save the information from that message and request a reservation, which results in the generation of the RESV message hop by hop back to the source. Whenever it receives another PATH message for that session requesting an increased rate, it will copy the information from the last reservation and again save the information from the current message. It only saves the last successful PATH request and the current PATH request. It will then generate a new RESV message. If this message results in RESV_ERROR, it will reissue a reservation request for the same rate as the previous successful reservation.
If the QoS Service Provider receives a RESV_ERROR for an aggregated RSVP reservation after attempting to increase the rate of the reservation, but the flow is still generating overlimit packets, it will create a filter that will mark the DSCP of that packet to a higher dropping level as well as set the ECN codepoint to Congestion Experienced (CE). The presence of CE in the ECN codepoint in packets arriving in a red network will indicate that the black side cannot reserve enough resources for the amount of RSVP traffic flowing between two red networks. It will be up to the red side to attempt to reduce the amount of traffic.

IV. EXPERIMENT AND EVALUATION

Once the QoS Service Provider and the RSVP daemon are started on all red clients and black routers, with the configuration files appropriate for their positions in the network, we can start an application, see Figure 4. First we start two server applications on red client 2. One is an RSVP receive application that tells the RSVPD that it will accept RSVP sessions on interface 1, port 5666, UDP protocol. The second is a network server that actually listens for UDP packets on port 5666 and just prints them to the screen.

We now run a test client on red client 1 that sends a certain number of UDP packets to red client 2. The client first sets the qinfo.qosmech field in the qos_info structure to INTSERV and calls the QSocket function for protocol SOCK_DGRAM. When QConnect is called, the QOSAPI sends this information to the QoS Service Provider. Since qinfo.qosmech is set to INTSERV, it will request an RSVP session from the RSVPD. The RSVPD will send an RSVP_PATH message to red client 2. As it is leaving red client 1, its DSCP will be set to AF42. The packet is forwarded to HAIPE device 1. Since it is from red network 1 to red network 2, HAIPE device 1 will set up an IPSec tunnel between itself an HAIPE device 2. Once the tunnel is established, HAIPE device 1 will encrypt the RSVP_PATH message in an ESP packet and forward it to black router 1. Note that the DSCP value in the RSVP_PATH message is copied to the ESP packet IP header. The kernel of black router 1 will notice that a packet from HAIPE device 1 with DSCP set to AF42 is being forwarded to its black-network interface and inform the QoS Service Provider. Upon receiving this notification, the QoS Service Provider will use the black-router-to-haipe-device table to determine the black router addresses that are connected to the source and destination IP addresses in the ESP packet. It will use them to request, through the RSVP API, an RSVP session between black router 1 and black router 3. The RSVPD will initiate an RSVP_PATH message to black router 3, port 9999, ESP protocol. Since the QoS Service Provider on black router 3 has started a receive API with those parameters, the RSVPD will forward the RSVP_PATH message to the QoS Service Provider. The QoS Service Provider will request an RSVP_RESV message to be sent hop-by-hop back to black router 1. When the RSVPD on black router 2 receives the RSVP_RESV message, it requests the scheduler for admission, which sends the request to the QoS Service Provider. If it has not allocated all of the bandwidth reserved for AF4, it will allow admission, which will result is a filter request. Since the destination port is 9999, the QoS Service Provider knows it is for aggregated RSVP, so it must consult the black-router-to-

haipe-device table to retrieve the IP address that will be in the ESP packet in order to create the TC U32 filter. This filter is created in the DSMARK qdisc and will map the ESP packets from HAIPE device 1 to HAIPE device 2 into AF41. The RSVP_RESV message is then sent to black router 1, where the same procedure will occur, but since this is the originating node the RESV event is returned to the QoS Service Provider, which now knows that the aggregated RSVP session is established. It will start monitoring the packets going through the newly created filter.

Meanwhile, the ESP packet will have traveled to HAIPE device 2, which will decrypt it and send it to red client 2. The RSVPD on red client 2 will send it to the RSVP receive API application, that we started earlier, which will request a reservation from the RSVPD. The RSVPD will initiate an RSVP_RESV message to red client 1, which will be forwarded to HAIPE device 2. It will encrypt the packet and send it out. It then flows back through the black routers and goes to HAIPE device 1, which decrypts and sends it to red client 1. The RSVPD on red client 1 will ask the QoS Service Provider for admission. If admitted, a filter request is sent to the QoS Service Provider, which will create a filter in the DSMARK qdisc for AF4 in order to map the packets for this flow into AF41. Since this is the originating node for the red RSVP session, the QOSAPI is notified that the RESV was received and it sets up a filter in the HTB qdisc to map this flow to the DSMARK qdisc for AF4.

Now, when the application on red client 1 begins to send its packets, the filter in the HTB qdisc sends this packet to the DSMARK qdisc for AF4. As long as the flow stays within the original flow spec request, the filter in the DSMARK qdisc will mark the packets with the DSCP for AF41. When the packets reach HAIPE device 1, they are encrypted in ESP packets and sent to black router 1. On black router 1, since the DSCP is set to AF41, the packets are sent to the DSMARK qdisc for AF4, where the filter for that flow will again mark the DSCP as AF41, as long as the flow stays within the original flow spec request. The packets will flow through the black network with a DSCP of AF41 as long as the flow stays within the original flow spec request. When they reach HAIPE device 2, they are decrypted and sent to red client 1 and eventually to the test server application that we started earlier, which will print the original message to the screen.

Meanwhile, the QoS Service Provider on black router 1 is monitoring the flow through the filter for that flow in the DSMARK qdisc for AF4. If there are no packets through that filter for 100 seconds, the QoS Service Provider will request to release the aggregated RSVP session, which will result in the RSVPD sending an RSVP_PATH_TEAR message to black router 3 and all allocations and filters are released along the black network. The QoS Service Provider on black router 1 will reset the monitoring in the kernel for DSCP of AF41, AF42, and AF43 from HAIPE device 1.

V. SUMMARY

We have developed and demonstrated a Network QoS Management framework that addresses the objective of provisioning IntServ across cryptographic boundaries within the Linux operating system environment. The Network QoS Management framework includes DiffServ adaptation to dynamic link states, QoS Service Provider, QoS API for QoS-aware applications, and incorporation of RSVP-based IntServ functionality. We have also extended the IntServ framework to support aggregation of RSVP-based IntServ sessions in HAIPIS environments with dynamic resizing capability. Finally, we have extended the implementation of QoS provisioning via RSVP-based IntServ across HAIPE boundaries to support interchange of control signals, using the ECN bypass, for session error handling and reporting functionality.

As a next step, we plan to build upon our QoS provisioning via RSVP-based IntServ across HAIPE boundaries by adding the capability for error handling and resource allocation resizing based on bandwidth estimations of dynamic link states within the network.

ACKNOWLEDGEMENTS

This work was supported by Boeing Phantom Works (PW) Network Centric Operations (NCO) Thrust.

REFERENCES

1. A. Ayyagari, O. Brewer, T. Griem, J. Kim, "DiffServ Extensions to Dynamic Link States and Bandwidths for QoS-Aware Mobile Networking Applications," MILCOM 2005, October, 2005.
2. R. Braden, D. Clark, S. Shenker, "Integrated Services in the Internet Architecture: an Overview," IETF RFC 1633, June 1994.
3. J. Wroclawski, "The Use of RSVP with IETF Integrated Services," IETF RFC 2210, September 1997.
4. Y. Bernet, P. Ford, R. Yavatkar, F. Baker, L. Zhang, M. Speer, R. Braden, B. Davie, J. Wroclawski, E. Felstaine, "A Framework for Integrated Services Operation over Diffserv Networks," IETF RFC 2998, November 2000.
5. KOM RSVP Engine, http://www.kom.tu-darmstadt.de/rsvp/.
6. J. Wroclawski, "Specification of the Controlled-Load Network Element Service," IETF RFC 2211, September 1997.
7. A. Terzis, J. Krawczyk, J. Wroclawski, L. Zhang, "RSVP Operation Over IP Tunnels," IETF RFC 2746, January 2000.
8. B. Gleeson, A. Lin, J. Heinanen, G. Armitage, A. Malis, "A Framework for IP based Virtual Private Networks," IETF RFC 2764, February 2000
9. F. Baker, C. Iturralde, F. Le Faucheur, B. Davie, "Aggregation of RSVP for IPv4 and IPv6 Reservations," IETF RFC 3175, September 2001