Notes on Jrsvp --

RSVP Implementation in Java

Bob Braden, Jeff Kann, Graham Phillips, Ya Xu, and Alberto Cerpa

August 1999
USC Information Sciences Institute
Valid for ASP 1.5

1. INTRODUCTION

These notes provide an introduction to Jrsvp, a reference implementation of RSVP written in Java. This code was developed at ISI under the ARP (Active Reservation Protocol") project, a DARPA-funded research effort in active networking.

RSVP is an Internet signaling protocol that was developed to provide end-to-end QoS signaling for Internet integrated services. RSVP is distinguished by its support for IP multicast data meshes, by its receiver orientation, and by its use of soft state.

ISI previously developed rsvpd, a "C" implementation of RSVP. Rsvpd has been very widely distributed, and it has been ported into a variety of environments. We have now re-implemented most of RSVP in Java as a program called Jrsvp. We believe that Jrsvp will be more useful than rsvpd as a reference implementation of RSVP, for several reasons.

(1) The object-orientation of Java should make the structure and function of the code more transparent, so Jrsvp will more effectively document a set of algorithms that realize the RSVP protocol.

(2) Certain features of Java, especially garbage collection, add to the simplicity and understandability of Jrsvp compared to rsvpd.

(3) The platform-independence of Java should be a major advantage in porting Jrsvp onto a new platform. The basic requirement will be a JVfor the target system.

(4) Because of its relative clarity and simplicity, the Java version will make a much more suitable platform than the "C" version for experimenting with future extensions of the RSVP protocol.

2. CLASS STRUCTURE

The structure of an object-oriented program is essentially defined by its class structure. The reference version of Jrsvp consists of classes in three packages:

rsvp package: All RSVP-specific classes
asp package: Contains control and system routines
lib package: Contains useful library routines

This section summarizes the classes in these packages that are used by the reference implementation. We omit the .java suffix on Java class names, assuming the rsvp package unless an explicit package qualification is included.

2.1 RSVP Protocol Classes

These classes in the rsvp package implement the RSVP protocol.

2.1.1 A class for each RSVP Object type

RSVP's network encoding [RFC 2205] uses strongly typed data objects, typed by a pair of octets: ("class number", "C-Type"). Most of these RSVP object classes are represented within Jrsvp by corresponding Java classes, subclasses of the abstract base class RsvpObject. Each of the subclasses encapsulates the structure and contents of a particular RSVP data object using the data structures natural to Java.

For example, subclasses of RsvpObject with the following names mirror names in the RSVP spec:

- Session
- SenderTspec
- SenderTemplate (A)
- Adspec
- Style (A)
- Scope
- Flowspec (A)
- Filterspec (A)
- Errspec
- Confirm

Those names labelled with "(A)" are abstract classes that have multiple non-abstract sub-classes for different semantic variations. Note that these variations do not necessarily correspond to the RSVP C-Types, which may define variations that differ only in syntax. For example, the Java objects encode IPv4 and IPv6 addresses internally as just byte arrays, so the IPv4/IPv6 distinction does not appear in the Java class structure. On the other hand, the different RSVP styles share a single C-Type in RSVP but are distinguished by different subclasses in Java.

The full hierarchy for these classes representing RSVP objects is as follows:

- RsvpObject
  - Session
  - IntServSpec (A)
    - Adspec
    - Flowspec (A)
      - SenderTspec
      - Flowspec_G // Guaranteed
      - Flowspec_CL // Controlled-Load
  - Style (A)
    - Style_WF

```
 o Style_FF
 o Style_SE

o Filterspec (A)
 o SenderTemplate (A)
  o Filterspec_n // Normal FilterSpec
  o Filterspec_g // GPI FilterSpec
o Errspec

o Confirm

o Scope

o Challenge

o Integrity

o Diagnostic

o DiagResp

       o DiagRoute

o DiagSelect
```

These classes generally implement the following constructor:

```
 public <ClassName>( buf[] msg, int offset )
```

This de-serializing constructor parses the external RSVP protocol representation of the object and constructs the Java object. The RSVP protocol object, which is self-defining, begins at the specified offset in the given byte array. A few RSVP objects (TIME_VALUES and Style_xx) are exceptions and do not have such a constructor.

These classes also implement the following abstract methods of RsvpObject:

```
(1) A serializer method to convert the object into its external
    RSVP protocol representation in a DataOutputStream.

 public void writeProto( DataOutputStream dout )

(2) An "equals()" method.

(3) A "toString()" method for debug printout.
```

Note that writeProto() is analogous to the writeObject() method of the JDK serialization mechanism. However, istis somewhat different -- writeProto (de-)serializes only the protocol data, not the entire class -- so it is given a somewhat different name. It would be possible to replace the de-serializing constructors with analogous methods of the form readProto(DataInputStream din); this is left as a future exercise in abject consistency.

Specific RVSP classes have sub-hierarchies appropriate to the semantics of the objects; see the list above.

2.1.2 A class for each RSVP message type

An RSVP message is represented internally by the RsvpMsg abstract class, with subclasses for each RSVP message type:

- PathMsg
- PathTearMsg
- PathErrMsg
- ResvMsg
- ResvTearMsg
- ResvErrMsg
- ConfirmMsg
- ChallengeMsg
- ResponseMsg
- DiagMsg (A)
  - DreqMsg
  - DrepMsg

An instance of one of these RsvpMsg classes is created for every message that is received or sent by Jrsvp. Its fields contain an internal-format representation of the RSVP message. This representation consists of values as well as references to Java objects that represent individual RSVP objects (see 2.1.1).

Each RsvpMsg class has a constructor of the form:

```
public <xxxxMsg>( Session sess)
```

The RsvpMsg classes implement the following abstract methods of RsvpMsg:

(1) void writeProto(DataOutputStream dout) throws IOException

A method to serialize the message, i.e., encode its internal format to (re-)create the body of an outgoing RSVP message of this type. (The IOException should never be thrown (?)).

(2) void process()

Process this RSVP message upon receipt. These methods correspond to the xxx_accept calls of the C version.

(3) A "toString()" method for debug logging.

2.1.3 A Class for an RSVP Protocol Message

The RsvpProtoMsg class defines an RSVP protocol message in external format, i.e., as a byte array. It has the following constructors:

RsvpProtoMsg(RsvpMsg)

RsvpProtoMsg(buf[], int)

The first constructor creates an external format message from the given internal message, for transmission across the network. This includes building the common header, invoking the writeProto()

method of the RsvpMsg class to build the body, and computing the checksum or Integrity object. The second constructor form creates an external format message received from the network.

This class exports the following methods.

void send_out_vif(AddressNet, Interface, int TTL)

Send this message to a specified destination through a specified interface and with a specified TTL.

void send_out(AddressNet)

Send this message to a specified destination.

RsvpMsg getRsvpMsg()

Build an RsvpMsg object from this RSVP protocol message. This includes parsing the Common Header, checking the checksum, constructing an appropriate RsvpMsg object, and invoking the readProto() method of that object to parse and check the rest of the message.

2.1.4 A class for each element of RSVP state

For each element of RSVP state, there is an instance of a state class. The state class names are generally chosen to match the terminology of RSVP documentation (and of rsvpd):

- PSB (Element of Path state)
- RSB (Element of Reservation state)
- MRS (Element of merged reservation state)
- JrapiSession (Element of API session state)

Jrsvp algorithms differ in one significant way from those of rsvpd. Rsvpd deferred the merged of reservation state created until it needed to send reservation refresh messages; in contrast, Jrsvp explicitly maintains the merged state at all times. This merged state is kept in MRS instances. The purpose of maintaining the MRS state is to make more explicit the state consistency rules of RSVP. We believe that the algorithms of "lazy merging" used by rsvpd can be derived from the MRS rules built into Jrsvp.

2.1.5 Code-only Classes

The classes we have described so far may be characterized as "data-bearing"; i.e., each instance contains some data structure and directly-related methods. However, there are many sub-algorithms required by RSVP that do not fit neatly into an object-oriented framework, since they cut across multiple data-bearing classes.

Jrsvp groups these cross-cutting methods into two "code-only" classes, as follows.

o RsvpResv

This class contains many methods for establishing and maintaining state consistency.

o RsvpAPI

This class contains methods for converting between the ("RAPI") API interface implemented by rsvpd and a new internal Java-oriented API interface. This allows an existing C application using RAPI to use Jrsvp interchangeably with rsvpd.

[It should be possible to get rid of RsvpAPI, by folding all the API-specific code into the rest of the processing. It is unclear whether this would enhance or diminish the clarity and maintainability of the code.

It complicates serialization, since there are now two different external representations.

Easy to make RapiProtoMsg parallel to RsvpProtoMsg class, and it would map RAPI requests into (partial) PathMsg and ResvMsg objects. But common parent RsvpMsg would have to include state specifying which external representation is in use, to choose the right alternative code in readProto() and writeProto() methods.

Must add analogous state to Rsvp object classses.
]

A standard Java programming style would apply the "static" attribute to all the methods in these code-only classes. As a result, no instances of these code-only classes would ever need to be instantiated. However, ISI's development of this reference implementation of RSVP is the first step in a larger research agenda, which will allow dynamic loading of modified signaling algorithms across the network. This dynamic update mechanism will depend upon class inheritence. Static methods are unsuitable for this purpose since they cannot be overridden. This explains the one anomaly of Jrsvp: it contains only a few trivial static methods.

As a result, Jrsvp constructs one instance of each of the code-only classes at initialization time, and references to these instances are available as static fields in RsvpMain.java:

```
public static RsvpResv  resvObj;
public static RsvpAPI  APIObj;
```

2.2 State Storage Classes

2.2.1 SessionBase class

As a signaling protocol, RSVP builds state on behalf of independent units of activity called "sessions". For each active session, Jrsvp constructs an instance of the SessionBase class. Jrsvp builds a hash table to locate SessionBase objects efficently. Each SessionBase object contains the head of lists

of state objects relating to that session: PSBs, RSBs, and JrapiSession objects. It also contains a reference to the Session object that contains the actual data that defines the RSVP session: the destination IP address, destination port, and IP protocol Id.

2.2.2 Soft-State Container classes

The asp package provides RSVP with a container class with soft-state timing functions. See description in PPI spec.

- asp.StateContainer

The following adaptation and key classes are used for storing and retrieving RSVP path and reservation state:

- asp.ContainerAdapter (I)
  PsbAdapter, RsbAdapter

- PsbKey, RsbKey, PsbKeyBad, RsbKeyBad

2.3 Control Classes

2.3.1 asp.AspMain class

This class contains the entry point for all versions of Jrsvp. It also contains general initialization code and command-line parsing code.

2.3.2 RsvpMain class

RsvpMain is contains a secondary collection of control functions, which is more RSVP-specific than the functions in asp.AspMain. Its main functions include parsing the Rsvp configuration file, performing RSVP-specific initialization, and sending RSVP messages to the network.

2.3.3 RsvpBase class

This class performs an interface function. It implements the upcall method receivePacket(), which the asp package uses to pass received network and API packets for RSVP processing. It also contains some "proxy constructor" methods whose function is described elsewhere.

2.4 Network I/O and Routing Classes

2.4.1 asp.NetIO, asp.NetIONative classes

The NetIO class provides functionality to send and receive IPv4 and IPv6 packets. The NetIO class was written to supply network I/O capabilities that RSVP needs that are not supplied by the JDK. The missing features include not only common capabilities such as raw I/O, but also unusual capabilities such as the ability to send a multicast packet only to a specific interface. A large portion of the NetIO class is implemented using native methods, using C code taken from rsvpd, ISI's C version of RSVP. NetIO currently works on Solaris, FreeBSD, and Linux.

2.4.2 Interface Classes

These classes are used to represent the communication ports through which a communication node sends and receives packets. These ports, which are customarily called "[network] interfaces", should not be confused with the interface entity in the Java language.

o asp.InterfaceNet (A),
o asp.InterfacePhy
o asp.InterfaceVir
o asp.InterfaceAPI
o asp.InterfaceAPIApp
o asp.InterfaceWildCard

Classes InterfacePhy and InterfaceVir represent physical and virtual (mrouted) interfaces. InterfaceAPI represents the local API. For each API session (corresponding to a TCP connection), an instance of the runnable subclass InterfaceAPIApp is constructed and a thread spawned. The class asp.InterfaceWildCard is used for sending packets without specifying which interface to use.

o RsvpInterfaceNet (A)
o RsvpInterfacePhy
o RsvpInterfaceVir
o RsvpInterfaceAPI

Each RsvpInterface*** object defines a real, virtual, or pseudo interface for sending and receiving data packets. RsvpInterfaceAPI is used as a marker in the if_vec table to represent a pseudo-interface corresponding to all API sessions. These RSVP-specific interface classes inherit general interface parameters from asp.InterfaceNetIO and also contain some RSVP-specific fields.

At initialization time, Jrsvp builds if_vec[], a vector of RsvpInterfaceNet objects that shadows an interface table in the asp package.

2.4.3 IP addresses

The lib.IPaddr class contains static methods that implement utility functions for dealing with IP addresses.

2.4.4 Classes for RSRR interface to routing

o asp.RSRRupCall (I), RsvpRsrr

See [PPI document]

2.5 Classses for Traffic Control

These classes provide an interface to kernel traffic control mechanisms: admission control, packet classifying, and packet scheduling.

At the top level, there is a "link-layer-dependent adaptation layer" or LLDAL [see RFC 2205]. This layer contains a set of methods that do processing that is specific to a particular link layer technology. The

set of required methods is defined by the (Java) interface class "LLDAL", and it subclasses actually implement the methods.

- LLDAL (I)
  - LL_KernTC

    This class implements traffic control for point-to-point network connections. There will be an instance of this class for each point-to-point network interface.

  - LL_API

    This class implements traffic control for the "API" virtual interface; these operations are currently null, although they could in principle control local end-host scheduling functions.

    The LLDAL classes in turn invoke methods in asp.TrafficControl to request traffic control operations (e.g., addFlowspec, modFlowspec, addFilter, etc.) for specific network interfaces. These calls go through intermediate methods in the TC class that serialize the Java variables into byte arrays.

- TC, asp.TrafficControl

  The asp.TrafficControl methods invoke a set of native methods that were developed to interface to the ALTQ scheduling package.

3. THREAD STRUCTURE

3.1. Main thread

The runnable class asp.ChannelRsvpLeg creates a thread for reading "legacy" RSVP packets from network interfaces and processing them. It receives using asp.NetIO and passes the byte array to the receivePacket method of the RsvpBase object.

3.2. API Thread and sub-threads

The runnable class asp.ChannelAPI opens a TCP socket for the API and listens on a particular TCP port (constant AA_API_DEMUX_TCP_PORT). When an application connects to this port, the asp.ChannelAPI thread spawns a new thread executing the runnable class asp.InterfaceAPIApp to service this connection.

The asp.InterfaceAPIApp object reads API requests from the TCP connection and processes them. This class also overrides the send method inherited from asp.InterfaceNet, in order to send an API response message back through the TCP connection to the same application that made the request.

The C version of RSVP uses a Unix socket for interprocess communication between a set of API library routines (rapi_lib, etc.) that are linked with an application, and the daemon. The JVM does not support Unix sockets (presumably because they are too system-dependent). We chose to modify convert the API library routines to use looped-back TCP connections to communicate with the Jrsvp daemon.

3.3. RSRR upcall thread

The Thread class asp.RSRRThread implements a thread to dispatch RSRR upcalls.

3.4. State container (timer) thread

The runnable class EManager (in asp/StateContainer.java) creates a timer queue manager thread for the StateContainer.

4. OPERATION

4.1 Receiving an RSVP message

4.2 Sending an RSVP message

4.3

...

5. TESTING

The distribution includes a test driver program called "rtap" that can be compiled and linked with the TCP version of IPC. This provides a convenient testing vehicle for Jrsvp.

6. NATIVE METHODS

The following classes include native methods in their implementation:

- lib.Debug
- lib.IPaddr
- asp.NetIO
- asp.RSRR
- asp.TrafficControl

All native methods comply to Sun's JNI (Java Native Interface). Therefore, this distribution of Jrsvp requires that the JVM support Sun's JNI.