NAME

rsvpd − Resource ReSerVations Protocol daemon

SYNOPSIS

rsvpd [**−D**] [**−d** *debug_bits*] [**−l** *logging_level*] [**−R** *router_addr*] [**−t** *mstat_TTL*]

DESCRIPTION

Rsvpd is a daemon program that uses the RSVP resource reservation protocol to set up reservation state in hosts and routers. It supports an API that allows applications to make reservation requests. It includes an adaptation module to the admission control and traffic control mechanisms provided by the output device drivers in the kernel.

OPTIONS

−D Execute rsvpd in debugging mode, i.e., do not detatch the process, and print debugging information to stderr.

−d Set the debugging mask to the integer *debug_bits*. This mask selects which additional information, if any, will be logged when the logging level (see below) is at least LOG_DEBUG. See the section below on LOGGING CONTROLS.

−l Set the logging level, which controls logging of data for debugging and diagnosis, to the integer *logging_level*. See the section below on LOGGING CONTROLS.

−R Specify *rsvp_router* as the name or dotted-decimal number of a first-hop router to be used by *rsvpd* running on a host. With this parameter, *rsvpd* will send a unicast copy of any UDP-encapsulated RSVP message directly to *rsvp_router*, in addition to copies it normally sends. The **−R** parameters is required only when the multicast TTL threshold of a tunnel or firewall would otherwise make the first-hop router inaccessible from the host.

−t Set the IP TTL value for multicasting diagnostic information (summary of the state) to *mcast_TTL*. This information is multicast when the DEBUG_MCAST_STATE bit (see LOGGING CONTROLS) is set in the debugging bit mask. The default is TTL = 1, i.e., one hop.

CONFIGURATION FILE

The RSVP daemon looks for a configuration file named *rsvpd.conf* in the current directory. This file may contain configuration commands, one command per line. Each command consists of a series of keywords separated by blanks or tabs; some keywords are followed by one or two parameters. The first keyword in the line is the name of the command; other keywords may come in any order. A blank line or a comment line beginning with '#' will be ignored.

There are currently two commands defined:

interface *iface_name* [police] [udpencap] [udpttl *nn*]

[refresh *rr*] [disable]

[integrity] [sendkey *id key*]

recvkey *id key*

Here *iface_name* is the name of a physical interface (e.g., 'le0'), to which the following keywords apply. More than one command line may be given for the same interface, and corresponding lines have a cumulative effect.

The following keywords are defined for the **interface** command:

police Traffic control policing is to be applied to the specified interface.

udpencap
Force *rsvpd* to use UDP encapsulation of RSVP messages on the specified interface. In most cases *rsvpd* will automatically configure itself to do UDP encapsulation on any interfaces on which it is required. The **udp** keyword should be required only by a router connected to a LAN which has no hosts that act as RSVP senders.

udpttl ttl
UDP encapsulation using the specified TTL is to be performed on RSVP messages sent out the specified interface. This keyword implies the **udp** keyword. The default TTL for encapsulation is 1.

The **udpttl** keyword is required only when the local environment includes RSVP-capable hosts separated by non-RSVP-capable routers, or to satisfy TTL threshold requirements. The TTL value must not exceed the hop distance to the first-hop router; violation of this restriction will cause gratuitous UDP encapsulation between routers.

disable Disable RSVP on the specified interface. [Not supported yet].

refresh *time*
Override the default refresh period for the specified interface. Here *time* is a new refresh period in seconds. [Not supported yet].

integrity
Integrity checking is required on messages received on the specified interface.

sendkey *id key*

Here *id* is a key id and *key* is a corresponding key for sending messages to the specified interface. *Id* and *key* must be 8 and 16 bytes respectively, written in hexadecimal notation, with no spaces.

recvkey *id key*

Here *id* is a key id and *key* is a corresponding key for receiving messages. *Id* and *Key* must be 8 and 16 bytes respectively, written in hexadecimal notation, with no spaces.

Here is an example configuration file

```
# rsvpd configuration
interface le0   udpttl 3   integrity refresh 60
interface le0  sendkey 24d12409e9a03a00  c0640a4abda195de6062afe2de5a7e47
interface le0  sendkey 24d12409e9a03a01  7fa12385f3ac29b333715ff314d56fc9
recvkey 14a2b34ec2b23377  32fc719d796f2ad764f36cf072dfc5d4
recvkey 14a2b34ec2b23378  68fa01780355d7772997e5bf92927985
```

LOGGING CONTROLS

The RSVP daemon logs state and event information for management, diagnosis, and debugging. The logged data is written into an rsvpd log file (e.g., */var/tmp/rsvpd.log*). It also appears on the console (stderr) if *rsvpd* is executed in non-daemon mode (i.e., with the **–D** flag). The rsvpd log file can grow to a maximum size of approximately 400KB; it will then be closed and renamed to *rsvpd.log.prev*, and a new log file will be started.

Logging is controlled by two integer parameters, the ‘debugging mask’ and the ‘logging level’. These parameters may be set on the **rsvpd** command line or dynamically using the **rtap** console interface (see rtap(8)).

Each log message has a priority, and it will appear in the log if its priority is at least equal to the logging level. The priorities (defined in <*sys/syslog.h*>) used by *rsvpd* are as follows:

3 LOG_ERR
These messages indicate system errors, configuration errors, internal logical errors within rsvpd, or logical errors in the client end of an API socket. They should never occur in normal operation.

4 LOG_WARNING
These messages indicate temporary resource shortage or protocol errors in RSVP messages received from remote hosts.

6 LOG_INFO
These message log changes of management parameters.

7 LOG_DEBUG
These message contain debugging information. This will generally consist of a one-line summary of the event. Then additional information may follow, depending upon the setting of the debugging mask bits DEBUG_IO and DEBUG_EVENTS.

8 LOG_HEXD
Dump all RSVP messages sent and received in hex.

For example, a logging_level of LOG_DEBUG will cause all events that occur to be logged, while a logging_level of LOG_INFO will log everything but debug messages.

The debugging mask is considered to be a set of bits; the bits and their symbolic designations in the code are as follows:

1 DEBUG_IO
If this bit is included, the contents of the each RSVP message will be formatted to the log following its transmission or receipt.

2 DEBUG_DS_DUMP
If this bit is included, the complete path and reservation state will be written into the log, periodically and when the state changes.

4 DEBUG_EVENTS
If this bit is included, additional details on API and kernel scheduling events will be logged following the corresponding event lines.

8 DEBUG_ROUTE
If this bit is included, a great deal of information concerning route lookups will be logged.

16 DEBUG_MCAST_STATE
This bit does not control logging. If it is on, *rsvpd* will multicast its internal state periodically. The RSVP tool *rsvpeep* will receive and format this information. The multicast TTL may be set using the **–t** parameter (see above).

32 DEBUG_TIMERS
If this bit is included, a great deal of information concerning the timer queue will be logged.

SEE ALSO

rtap(8), rstat(8), rsvpeep(8)

BUGS