The VRRP Project
(Virtual Router Redundancy Protocol)
Design Document

Revision 0.9.4
January 06, 2009

vrrp-core@sun.com

Table of content

1. Introduction

This project is to implement the VRRP protocol (Virtual Router Redundancy Protocol) version 3 for IPv4 and IPv6 (draft-ietf-vrrp-unified-spec-02), which is based on RFC 3768, VRRP version 2 for IPv4, with the goal of providing High Availability on Solaris. The project provides administrative tool as well as API interfaces for third-party software to manipulate the VRRP service.

VRRP specifies an election protocol that dynamically assigns responsibility for a virtual router to one of the VRRP routers on a LAN. The VRRP router controlling the IPv4 or IPv6 address(es) associated with a virtual router is called the Master, and forwards packets sent to these IP addresses. The election process provides dynamic fail over in the forwarding responsibility should the Master become unavailable. The protocol is designed to eliminate the single point of failure inherent in the static default routed environment. The advantage gained from using VRRP is a higher availability default path without requiring configuration of dynamic routing or router discovery protocols on every end-host.

2. Requirements

Key requirements for the VRRP implementation on Solaris are outlined below:

- Conforming to the standard and full implementation of required features of the VRRP protocol
- Extension to the standard for use on non-router appliances
- Convenient administrative interface and explicit APIs to deploy the service
- Minimize the processing overhead and complexity
- Independence of hardware (NICs)
- Minimize the influence to other network services and complete restoration for system configuration/status while VRRP objects are on/off or doing fail over
- Capabilities of protecting IP address(es) on vnics and link aggregations
- Support multiple MAC addresses (Virtual Router MACs for different instances) on a link

3. Architectural Overview

3.1 VRRP instance

As specified in the RFC, VRRP specifies an election protocol that dynamically assigns responsibility for a virtual router to one of the VRRP routers on a LAN. The VRRP protocol runs on each VRRP router, to keep their states.

In this project, the term "VRRP protocol instance" or "VRRP instance" refers to the VRRP protocol that runs on one VRRP router that belongs to a VRRP virtual router. A host can have multiple VRRP instances, in which case it belongs to more than one VRRP virtual routers.

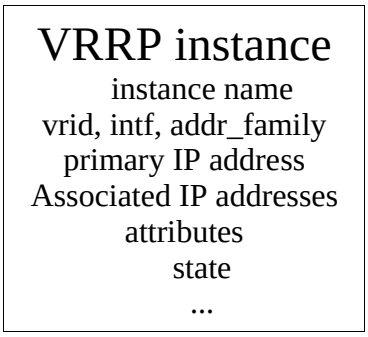

Fig 3.1

A VRRP instance consists of:

- The instance name (identifier, system-wide unique)
- VRID, interface name, address family
- Primary IP used as the source of the advertisement
- Associated IPs to be protected
- Attributes: priority, advertise interval, preempt mode, accept mode
- State information and statistics

3.2 Components of the VRRP implementation

The Components of the VRRP implementation include:

- An administrative tool vrrpadm to manipulate the service
- A daemon vrrpd controlled by the VRRP SMF service running on the background
- An exported library libvrrp for third party programs to exploit

- A repository vrrpd.conf for persistent configuration

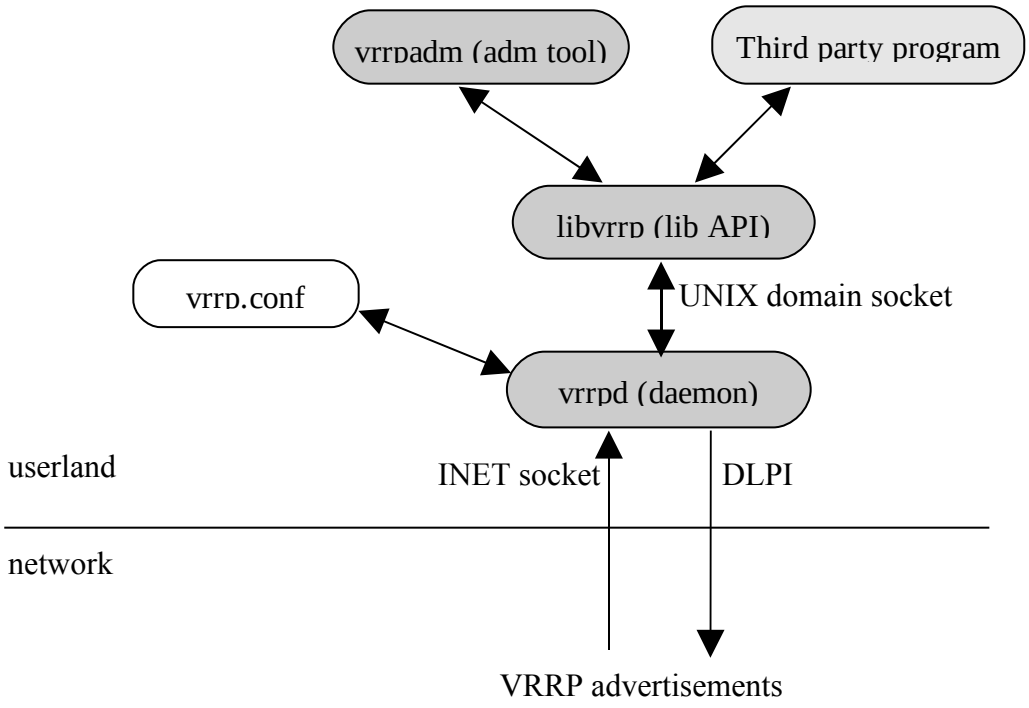

Figure 3.2

Each host running VRRP service will have the vrrpd daemon running on it. The daemon maintains all VRRP instances on this host. The configuration file defines one or more VRRP instances to be created when the vrrpd daemon is started. Later more VRRP instances can be added, either using the vrrpadm tool or using the libvrrp API calls. Manual edition to the configuration file is not recommended. Refer to Appendix B for the format and example of the configuration file.

The default configuration file located at `/etc/vrrpd.conf` will be used. The configuration file could be empty or not present, in which case no VRRP instance will be created.

The vrrpd daemon is under SMF control. The corresponding SMF service is:

fmri	svc:/network/vrrp:default
name	Virtual Router Redundancy Protocol daemon
dependency	require_all/none svc:/network/routing/ndp

The service depends on in.ndpd for Router Advertisements. The router administrator should manage the Router Advertisement configuration and

guarantee that the configurations are the same for VRRP on the master and the backup.

The pathname of the configuration file is set as property of the SMF service. On start/restart the service calls vrrpd and the configuration in the file is loaded. After vrrpd is started, the vrrpadm tool could be used to configure VRRP. The tool communicates with vrrpd through UNIX domain socket. Using vrrpadm, users can create, modify, destroy, send startup/shutdown events to, VRRP instances.

The vrrpadm tool exploits the libvrrp library, which exports a set of APIs to perform VRRP related operations. This library will be public so that third-party softwares can also use it to interact with VRRP service or write their own VRRP administration tools.

3.3 The vrrpadm Tool and Basic Operations

The vrrpadm tools is used for daily VRRP operations. Such operations are:

- creating a VRRP instance
- deleting a VRRP instance from the system
- modifying a VRRP instance's attributes
- sending a STARTUP or a SHUTDOWN event to a VRRP instance
- querying the status of the VRRP instance(s) in the system

Most of the operations can be performed using either a vrrpadm command or a corresponding libvrrp API. We'll discuss the vrrpadm tool in this section, and discuss the libvrrp in the next section.

The vrrpadm command looks like:

```
vrrpadm <subcmd> <options> <vname>
```

The sub-commands are: `create`, `delete`, `modify`, `startup`, `shutdown`, `show`, respectively.

Each command must specify the `<vname>`, which uniquely identifies the name of the VRRP instance to be operated on. The exception is the `show` command, which will show all VRRP instances in the system if `<vname>` is not specified.

The "`vrrpadm create`" command is used to create a VRRP instance.

```
vrrpadm create
```

```
-v <vrid> -i <intf> [-j <primary_addr>] -A <ipaddr_list>
[-p <priority>] [-d <adv_interval>] [-o <flags>]
[-P <protected_service>[,...]]
<vinst_name>
```

To uniquely identify the VRRP instance, the (vrid, intf, addr_family) three-tuple should be provided. The addr_family doesn't need to be provided though; it can be derived from `<primary_addr>`; if `<primary_addr>` is not provided, it can be derived from `<intf>`.

Note the the instance name must be provided. After an instance is created, its name is used in the operations that follows.

As discussed above, the VRRP instance can be used to protect a SMF service. In this case, the "`-P <protected_service>[,...]`" option is used to specify a set of SMF services/instances to be protected by this VRRP instance. For more discussion about using VRRP to protect SMF services, see Section 5.7.

The user then can use "`vrrpadm modify`" to modify the properties of an VRRP instance. Note only priority, adv_interval, preepmt mode, accept mode can be modified; other properties such as VRID or interfaces are considered "fatal" and shouldn't be modified.

```
vrrpadm modify
   [-p <priority>] [-d <adv_interval>] [-o <flags>]
   <vinst_name>
```

For a running VRRP instance, "`vrrpadm startup`" can be used to send a `STARTUP` event to it. On receiving this event, the VRRP instance, if in the `INITIALIZE` state, will leave the `INITIALIZE` state and eventually enter either `BACKUP` or `MASTER` state. Similarly, "`vrrpadm shutdown`" can be used to send a `SHUTDOWN` event; on receiving it, the VRRP instance, if in `BACKUP` or `MASTER` state, will enter the `INITIALIZE` state, which is actually a "paused" state.

```
vrrpadm startup <vinst_name>
vrrpadm shutdown <vinst_name>
```

The "`vrrpadm show`" command is used to show the information of one or more VRRP instances. As an exception, the `<vname>` can be omitted, and it'll show all VRRP instances in the system.

```
vrrpadm show [-p] [-o field[,...]] [<vinst_name>]
```

Finally, "vrrpadm delete" is used to delete a VRRP instance from the system.

```
vrrpadm delete <vinst_name>
```

These operations and changes are also synchronized to the configuration file immediately, and thus affect how the daemon works when it's restarted next time. Particularly, the result of "startup" and "shutdown" commands is recorded by the "active" field in the configuration file. If an instance is shut down, the corresponding "active" is set to FALSE, and next time when vrrpd is started, the "active" flag will cause the instance to stay in the INITIALIZE state.

Most of these commands (create, startup, shutdown, delete, modify) require the PRIV_SYS_NET_CONFIG and PRIV_NET_RAWACCESS privileges to operate; the show command doesn't require extra privileges.

For detailed description of the vrrpadm tool, refer to Appendix B.

3.4. The libvrrp Library and Basic Operations

This project will provide a set of APIs in libvrrp.so; Third party softwares can use these APIs to write their own tools to configure VRRP.

Almost each API has a corresponding vrrpadm sub-command, thus most of the basic operations mentioned in the previous section can be performed using either a vrrpadm command or a corresponding libvrrp API. The exception is the querying operations.

The APIs to do the basic operations are:

```
vrrp_create()
vrrp_destroy()
vrrp_setprop()
vrrp_startup()
vrrp_shutdown()

vrrp_query()
vrrp_lookup()
```

Each of the first five has a corresponding vrrpamd command. Vrrp_query() is used to get all information of a VRRP instance, similarly to "vrrpadm show <vname>". vrrp_lookup() is used to return the names of all VRRP instances

that match the given <vrid>, <intf>, <addr_family> combination.

Similarly to vrrpadm tools, the first five APIs require the PRIV_SYS_NET_CONFIG and PRIV_NET_RAWACCESS privileges; lacking of privileges will result in a VRRP_ENOPERM error. Vrrp_query() and vrrp_lookup() don't require extra privileges.

Similarly to the vrrpadm tool, most APIs requires the VRRP instance name as an argument.

The return value of the APIs are the vrrp_ret_t type. On success, it returns VRRP_SUCCESS; otherwise, it returns one of the error reasons:

```
typedef enum {
   VRRP_SUCCESS = 0,
   VRRP_EINVAL,        /* invalid parameter */
   VRRP_ENAMEINVAL,    /* invalid VRRP instance name */
   VRRP_EBADSTATE,     /* VRRP instance in bad state */
   VRRP_EVREXIST,      /* <vrid, intf, af> three-tuple exists */
   VRRP_EINSTEXIST,    /* vrrp instance name already exists */
   VRRP_ENOROUTE,      /* cannot determine interface by vi_pip */
   VRRP_EVRIP,         /* invalid virtual router ip */
   VRRP_ENOINST,       /* vrrp instance does not exist */
   VRRP_ENOPERM,       /* insufficient privilege */
   VRRP_ESYS           /* system error */
} vrrp_ret_t;
```

Note VRRP_ESYS indicates an error not specific to VRRP (for example, memory allocation failure), all others are VRRP-specific. The vrrp_strerror() function gives a human readable description of these errors.

For a detailed discussion, refer to Appendix A “Libvrrp API specifications”.

4. State transition and Fail-over

According to the RFC, a VRRP instance can have 3 states: INITIALIZE, BACKUP, MASTER.

4.1 manual startup or shutdown

When a VRRP instance is created, it may stay in the INITIALIZE state, if

"active" is set to false (see appendix for details). In this state, it just waits for the STARTUP event to start "running".

A STARTUP event can be sent explicitly to a VRRP instance that is in the INITIALIZE state using either "vrrpadm startup" sub-command or the vrrp_startup() API.

Similarly, a SHUTDOWN event can be sent explicitly to a VRRP instance that is in the BACKUP or MASTER state, using either the "vrrpadm shutdown" sub-command or the vrrp_shutdown() API.

The RFC clearly defines the INITIALIZE state and behaviors associated with this state. This state can be seen as a "paused" state for a VRRP instance. It is useful for diagnosis, or when VRRP instances are configured but not ready for running. A use case could be coordinating the start time for VRRP instances on multiple machines.

4.2 sub-state of INITIALIZE

There are two sub-states of the INITIALIZE state: shutdown and pending.

When a VRRP instance is specified to protect some SMF services/instances, it should not leave the INITIALIZE state on receiving a STARTUP event, until all of its protected SMF services/instances are in online state. If any of the protected services is in disabled/offline/maintenance state, the VRRP instance will stay in the INITIALIZE state. In this case, the VRRP instance is regarded as being in a “pending” sub-state, which indicates that it stays in INITIALIZE because some of its protected SMF services/instances are not online, and when they all come back to online state, the VRRP instance will leave the INITIALIZE state.

The sub-state “shutdown” indicates that the VRRP instances has received administrative “SHUTDOWN” events and will not come alive even when all of its protected SMF services/instances are online. It will only leave the INITIALIZE state when receiving administrative “STARTUP” events.

4.3 fail-over scenarios

The basic fail-over case for VRRP is when the Master host is down, or its network device fails to send/receive packets, the VRRP instance running on

the Backup host detects the absence of the VRRP advertisements and then take over the responsibility of the virtual addresses.

By sending a SHUTDOWN event to a VRRP instance in MASTER state manually, users could trigger intended fail-over.

Another scenario of fail-over is that the state changes of the protected SMF services/instances could trigger state transitions of the VRRP instances.

When an FMRI is specified for a created VRRP instance, the VRRP daemon will monitor the state changes of the according SMF service/instance and notify the VRRP daemon to examine the newest status of the SMF service/instance before making decision for state transitions.

For instance, a VRRP instance is running in MASTER state, with all of its protected SMF services in online state. When one of the protected SMF services turns into offline state at some point, the VRRP daemon gets notification, and change the state of the VRRP instance to INITIALIZE, with sub-state “pending”. This would trigger fail-over if there are Backup hosts running in the subnet.

When the offline SMF service comes back to online, the VRRP daemon will also get notification, and check that all protected SMF services are online, and then make the VRRP instance leave the INITIALIZE state. If the VRRP instance runs in preempt mode, this state transition will finally trigger preempted fail-over back.

As described above, by using the “-P” option with the create command, a VRRP instance could be used to provide redundancy for SMF services running on multiple hosts. The scheme is to trigger fail-over according to the health of the protected services.

Note that when a VRRP instance is in “shutdown” sub-state, it will ignore the notifications of the state changes of the protected SMF services/instances. Being in “shutdown” sub-state means the instance has received SHUTDOWN event and the administrator wants it to stay in INITIALZE state.

4.4 Sysevents Associated with State Transitions

When a VRRP instance transitions from one state to another, some programs may want to get notified so that they can do something. The sysevent

mechanism is to serve this purpose. When a state transition occurs, a system event that contains the VRRP instance name and the transition type (from what state to what state) will be delivered by the vrrpd daemon; interested programs can subscribe to these events to get notified.

The system event has the following properties:

class: `ES_vrrp`
subclass: `ESC_vrrp_state_change`
vendor: `SUNW`
publisher: `vrrpd`

An event has the following name-value pair attributes:

`vrrp_event_version` (integer): "1" for this version.
`vrrp_inst_name` (string): the VRRP instance name.
`vrrp_state` (integer): the current state.
`vrrp_prev_state` (integer): the previous state.

And

`vrrp_substate` (integer): the current sub-state.
`vrrp_prev_substate` (integer): the previous sub-state.

... if the corresponding state(s) is `INIT`. Note the 'substate' attribute(s) is/are not included if the corresponding 'state' attribute(s) is/are not `INIT`.

The value of the two 'state' attributes can be `NONE`, `INIT`, `BACK` or `MAST`. The value of the two 'substate' attributes can be `PENDING` or `SHUTDOWN`. State change from `NONE` means the VRRP instance was just created, and state change to `NONE` means the VRRP instance was just deleted.

The event subscriber, after receiving the events, can then call the `vrrp_query()` API to get more information about the VRRP instance.

With this solution, other services that use VRRP have a chance to do certain things when the VRRP instance state changes. There's another option, though, that the vrrpd daemon launch certain executables when the state transition occurs. It's described briefly below, but in this version we don't plan to implement it.

4.4.1. Considerations of External Executables Associated with State Transitions

Another possible solution is to run a pre-defined executable directly when a

state transition occurs.

With this solution, if an executable (binary or script), `/etc/vrrpd/sthook.<name>`, exists, the vrrpd daemon will automatically run the executable when the VRRP instance identified by `<name>` has a state transition. By default the system doesn't provide default executables. The user is responsible for putting files under `/etc/vrrpd/`.

This option is also simple, but has security issues. For example:
- Who will be the owner of these executables? What permissions will they have?
- Who has the permission to put files into /etc/vrrpd/?
- What priviledges will these executables have when they're running? (Because they're launched by vrrpd, which is running as root.)

These questions need careful examining in a real use case. Since no customer is asking for this feature, we decided not to use this option for now.

5. Implementation details

5.1 Communication between vrrpadm and vrrpd

When started, vrrpd creates the VRRP instances defined in the configuration file. The sysadmin can also use the vrrpadm tool to ask vrrpd to do certain operations: create, destroy, modify attributes, etc.

A set of private commands are defined between vrrpadm and vrrpd. When the user issues a vrrpadm command, say, "vrrpadm create" (to create an VRRP instance), the correpsonding private command (VRRP_CREATE_INST in this case) and its arguments are sent to vrrpd. Vrrpd then parses the command, performs the operation and returns the result to vrrpadm.

UNIX domain socket is used for communication between vrrpd and vrrpadm. Vrrpd creates an endpoint with a well-known address (/var/run/vrrpd.socket) so that vrrpadm knows where to send the messages.

5.2 Sending and receiving VRRP advertisements

VRRP instances in the master state send periodic advertisements as required

by the RFC specification. VRRP instances in the backup state receive advertisements and udpate their timer. To receive, raw IPv4 and IPv6 sockets are used. To send, DLPI in raw mode is used. The sockets and DLPI handle is associated with the interface that hosts the primary IP(v6) address. Differently from sockets, a single DLPI handle is used to send both IPv4 and IPv6 packets (actually raw Ethernet frames).

If multiple VRRP instances are created over the same network interface, they share the same set of sockets and DLPI handle.

5.3 Selection of primary IP address

As specified in RFC, when sending VRRP advertisements, the fields are set as:

src ip addr:
    IPv4: the primary IP address
    IPv6: the link-local address
dst ip addr: multicast VRRP address
    IPv4: 224.0.0.18
    IPv6: FF02:0:0:0:0:0:XXXX:XXXX (normally 0:12)
src mac addr: virtual mac addr of virtual router
    IPv4: 00-00-5E-00-01-<VRID>
    IPv6: 00-00-5E-00-02-<VRID>
dst mac addr: virtual mac addr of multicast VRRP address
    mapped from the dst ip addr

A primary IP address is required for IPv4 as specified in the RFC, for sending VRRP advertisement packets. It should be selected from the set of real interface addresses. Users could specify the primary IP using "-j <addr>" through CLI or "vrrp_inst_t.vi_ip" through API. If it is not specified by the user, the primary IP address will be chosen from the first address of the interface the VRRP instance resides.

For IPv6, the concept of "primary IP address" doesn't apply. The IPv6 version of VRRP just uses the link-local address to transmit the VRRP messages.

5.4 IPv6 link-local address

According to the RFC, the first IPv6 address of the VRRP protected addresses

must be link-local address. To conform the RFC, in the implementation the IPv6 link-local address will be automatically assigned for every created VRRP instance as the first protected IP address, which is formed by using the VRRP MAC to create the Modified EUI-64 identifier, as suggested by the RFC. It is used to transmit VRRP advertisement messages and Router Advertisements.

5.5 Implementation of virtual MAC addresses and IP addresses

As described in section 7.3, the virtual router is seen by the others by its associated IP and virtual MAC address. Only the Master responds to ARP requests or ND solicitations for the associated IP, and packets sending to the associated IP must be accepted (or forwarded) by the Master only.

If multiple virtual routers are configured on one interface, there could be multiple virtual MAC addresses residing on one interface. Thus simply changing the MAC address of the interface can't meet the requirement of implementing the virtual MAC. The Solaris VNIC technology will be utilized to implement the virtual MAC by creating one VNIC for each VRID.

According to the standard, in Accept_Mode the Master will accept packets designating the associated IP addresses, whether or not it is the owner. While when Accept_Mode is False, the Master must not accept those packets, although it must forward packets and respond to ARP requests or ND Neighbor Solicitations for the associated IP addresses, if it is not the owner. The defined behaviors could be summaries as in the following table:

	Accept packets	Forward packets	ND adv/ ARP response	RA adv
Master state Accept_Mode = True	Yes	Yes	Yes	Yes
Master state Accept_Mode = False	No	Yes	Yes	Yes
Backup state	No	No	No	No
Initialize state	No	No	No	No

In the implementation, for Master in Accept_Mode, the associated IP addresses will be configured on the VNIC, if it is not the owner. ARP response for IPv4, ND Neighbor Advertisements and Router Advertisements for IPv6 will work properly as required by the standard. For Backup in

Accept_Mode, the associated IP addresses will be configured on the VNIC, while the VNIC link will not be brought up. This configuration allows local applications bind to the associated IP addresses while at the same time the IP addresses are unseen by outside world.

For Master not in Accept_Mode, proxy ARP/ND cache will be created for the associated IP addresses, using the SIOCSXARP ioctl (equivalent to using "arp -s") and the SIOCLIFSETND ioctl for IPv4 and IPv6 respectively, instead of configuring the IP addresses directly on the VNIC. Thus packets designating the associated IP addresses will not be accepted by the Master. One exception is the IPv6 link-local address on the created VNIC. It is brought up with the VNIC and thus needs special scheme to prevent accepting packets. In the implementation a new interface flag IFF_LL_NOACCEPT is introduced to mark a VNIC as non-accept mode. On receiving ioctl SIOCSLIFFLAGS request for IFF_LL_NOACCEPT, the IRE entry for the link-local address of the interface will be marked as IRE_MARK_NOACCEPT. If an ire which is looked up for a link local destination address turns out to have this flag marked, the input packets will be dropped in ip_rput_data_v6().

If the Master is the owner, the associated IP is physically configured on the machine. According to the standard, it must respond to ARP requests or ND Neighbor Solicitations with virtual MAC addresses. To achieve this, the associated IP address on the physical interface will be unconfigured and configured on a created VNIC instead.

The implementation of associated IP in various state could be summarized as below:

VRRP	Associated IP	configuration	VNIC status
Master state Accept_Mode = True or Address owner	IPv4	On VNIC	Up
	IPv6 link-local	On VNIC	Up
	IPv6 non-link-local	On VNIC	Up
Backup state Accept_Mode = True	IPv4	On VNIC	Down
	IPv6 link-local	On VNIC	Down
	IPv6 non-link-local	On VNIC	Down
Master state Accept_Mode = False	IPv4	Proxy ARP entry	Up
	IPv6 link-local	On VNIC	Up; IFF_LL_NOACCEPT On

	IPv6 non-link-local	ND cache	Up
Backup state Accept_Mode = False	IPv4	N/A	Down
	IPv6 link-local	On VNIC	Down
	IPv6 non-link-local	N/A	Down
Initialize state sub-state = pending Accept_Mode = True or Address owner	IPv4	On VNIC	Down
	IPv6 link-local	On VNIC	Down
	IPv6 non-link-local	On VNIC	Down

5.6 Router Advertisements

The RFC is designed for routers and there are some clauses defining the Router Advertisement behaviors for VRRP IPv6 associated addresses.

- The master must send ND Router Advertisements for the virtual router while the backup must not.
- When fail over occurs the backup should take over the responsibility to send Router Advertisements using the same options as that of the original master.

The VRRP service relies on ndp service for sending Router Advertisements for the VRRP associated IPv6 dresses. The administrator should be responsible of managing in.ndpd in correct configuration. The RA options for the interfaces on which VRRP is running must be the same on master and backup.

To make VRRP work for non-router cases, the protocol is extended to allow violating the first clause. If in ndpd.conf the interface on which VRRP is running is specified not sending advertisements, the VRRP will not consider it is an error and will treat it as a non-router configuration.

5.7 SMF interaction

As discussed in section 3.3 and 4.3, VRRP can be used to protect other SMF

services by specifying one or more fmri when a VRRP instance is created using either "`vrrpadm create ... -P <fmri>[,...]`" or "`vrrp_create()`".

To monitor the state changes of a SMF service/instance which is to be protected by VRRP, the VRRP daemon utilize the SMF private interface `_scf_pg_wait()` to get notification when the group property "restarter/state" of the SMF service/instance is changed. For each SMF service/instance to be protected, a thread is created to be pending at `_scf_pg_wait()` to get state change notifications.

For a VRRP instance running in MASTER or BACKUP state, when one of its protected SMF services turns into offline/disabled/maintenance state, it will get notification, and then change its state to INITIALIZE, with sub-state "pending".

For a VRRP instance running in "pending" state, when all of its protected SMF services turn into online state, it will get notification and then change its state to BACKUP or MASTER.

When a VRRP instance is in INITIALIZE state with "shutdown" sub-state, it simply ignores state change notifications of SMF services.

vrrpd doesn't take care of the protected SMF services except monitoring their state changes. The user is responsible for making sure the SMF services are running normally after the VRRP instance is created.

Let's examine a special case for protecting services. Most network services are provided by applications which usually use bind() to provide service on either a well-known IP address, or bind to INADDR_ANY. For the former, the well-known IP address is the very IP address associated with (or protected by) this VRRP instance, and this IP address doesn't exist until VRRP sets it up. Thus, before the VRRP instance is created and running, the protected SMF service won't be online because it's unable to bind to its well-known IP address.

So, if VRRP is used to protect a SMF service that bind to a particular IP address, the action of enabling the SMF service should be taken after the creation of the VRRP instance. Otherwise the SMF service will be in maintenance state for failing to bind to the IP address that the VRRP instance will configure. The sequences would be:

1) The protected SMF service is in disabled state, or in maintenance state

for failing to bind to the IP address.

2) The VRRP instance gets created and started, but stays in INITIALIZE state with “pending” sub-state, because the protected SMF service is not online.
3) Enable or refresh the protected SMF service, and it turns into online state for binding to the IP address successfully.
4) The VRRP instance gets notification of the state change of the protected SMF service, and turns into MASTER state.

5.8 Timers and threading

Each VRRP instance in the MASTER state has a advertisement timer that fires to trigger sending of advertisement. Each VRRP instance in the BACKUP state has a Master down timer that fires when advertisement has not been heard for a specified period of time, to trigger state transition.

To minimize the complexity, the library libinetutil is utilized to maintain the timers and the socket calls. The processing of received commands and advertisements are serialized by poll(2) in libinetutil. Thus apart from the separate threads for SMF interaction, there is only one single thread that will be running for the vrrpd main routine.

For each SMF service/instance to be protected, a thread is created to be pending at `_scf_pg_wait()` to get state change notifications, and then send notifications to the main thread through the UNIX domain socket. These threads do not access any internal data, thus there is no lock needed for all the data structures in the implementation.

6. Limitations and interoperations with other network features

The VRRP network implementation utilizes the vnic technology to achieve applying multiple Virtual Router MAC addresses on an interface. The requirement of the ability to create vnics makes it unable to work in exclusive-zones.

VRRP service cannot be applied on IPMP links. That's because VRRP implementation requires manipulating VNIC links and mac addresses thus VRRP instances are applied on data-links, while IPMP works on IP.

VRRP service cannot work with DHCP. The protected IP address must be explicitly specified on creation.

VRRP service can work with link aggregation, i.e. VRRP instances could be configured on aggregated interfaces and work properly.

7. Security considerations

7.1 Privilege required for CLI/API users

The vrrpd daemon requires certain privileges for operation requests either from vrrpadm commands or from programs using libvrrp APIs. The daemon checks if the process of the requesting vrrpadm command has sufficient privilege to perform the requested actions. If the privileges are insufficient, the request will be rejected.

Only basic privilege is required for the "`vrrpadm show`" command and `vrrp_query()` and the `vrrp_lookup()` APIs. Other commands and APIs require `PRIV_SYS_NET_CONFIG` and `PRIV_NET_RAWACCESS` privileges.

7.2 Authentication and confidentiality

According to the RFC, VRRP for IPv4 and IPv6 does not currently include any type of authentication or confidentiality.

Appendix A. VRRP library Specifications

To maintain backwards compability, a macro `LIBVRRP_VERSION` is defined in this library so that unbundled software can still work when some interfaces are changed.

A.1 Error handling

The return value of the functions indicates whether the operation succeeded or not. All VRRP functions return `VRRP_SUCCESS` on success. Otherwise, a different return value is returned to indicate an error. The return values are defined in the following enum. Note `VRRP_ESYS` indicates an error not specific to VRRP (for example, memory allocation failure), all others are VRRP-specific.

```
typedef enum {
    VRRP_SUCCESS = 0,
    VRRP_EINVAL,          /* invalid parameter */
```

```
    VRRP_ENAMEINVAL,     /* invalid VRRP instance name */
    VRRP_EBADSTATE,      /* VRRP instance in bad state */
    VRRP_EVREXIST,       /* <vrid, intf, af> three-tuple exists */
    VRRP_EINSTEXIST,     /* vrrp instance name already exists */
    VRRP_ENOROUTE,       /* cannot determine interface by vi_pip */
    VRRP_EVRIP,          /* invalid virtual router ip */
    VRRP_ENOINST,        /* vrrp instance does not exist */
    VRRP_ENOPERM,        /* insufficient privilege */
    VRRP_ESYS            /* system error */
} vrrp_ret_t;
```

Processes using the APIs to manage VRRP service and configuration must have sufficient priviledges, otherwise VRRP_ENOPERM will be returned.

The `vrrp_strerror()` function gives a human readable description of these errors.

```
const char *vrrp_strerror(int err);
```

A.2 VRRP name manipulation

A VRRP instance is uniquely identified by an instance name. Most APIs that handles a VRRP instance have the name (`char *`)as the input parameter.

A valid VRRP instance name contains less than 16 characters which should be letters, digits, or underscores. Using a invalid instance name will cause some functions to return `VRRP_ENAMEINVAL`.

A.3 VRRP instance

A.3.1 Create a VRRP instance

```
vrrp_ret_t  vrrp_create(vrrp_inst_t *inst);
```

To create a VRRP instance, one should use a `vrrp_inst_t` structure containing all the required information to call `vrrp_create()`.

This function requires the `PRIV_SYS_NET_CONFIG` and `PRIV_NET_RAWACCESS` privileges.

The `vrrp_inst_t` structure contains necessary information for setting VRRP instance properties. Passing in invalid parameters would cause the function to return `VRRP_EINVAL`. If there is already an existing VRRP instance having the same virtual router ID, interface name and address family, the function will return `VRRP_EINSTEXIST`.

Instance names are globally unique on a system. Using a name belonging to an existing instance will cause the function to return `VRRP_EINSTEXIST`.

If the property vi_active is B_TRUE, the created VRRP instance will get a "Startup" event upon creation. Otherwise the created VRRP instance will be in INITIALIZE state.

```
typedef uint8_t      vrpri_t;      /* VR priority, 0-255 */
typedef uint8_t      vrid_t;       /* VRID, 0-255 */

enum {
    VRRP_STATE_INIT = 1,
    VRRP_STATE_MAST,
    VRRP_STATE_BACK,
    VRRP_SUBSTATE_SHUTDOWN,
    VRRP_SUBSTATE_PENDING
};

#define    VNSIZ             16
#define    VRRP_MAXLEN_FMRI  512

typedef struct vrrp_inst {
    char          vi_name[VNSIZ];      /* vrrp instance name */
    boolean_t     vi_active;           /* Start up? */

    vrid_t        vi_id;        /* VRID of this vr */
    int           vi_af;        /* IP addr family */
    vrrp_addr_t   vi_pip;       /* primary IP */
    uint8_t       vi_ipnum;     /* number of virtual router IP, 1-255 */
    void          *vi_ip;       /* pointer to the virtual IP array */
    char          vi_ifname[LIFNAMSIZ];

    vrpri_t       vi_prio;      /* priority */
    int           vi_adv_intv;  /* advertisement interval */
    boolean_t     vi_preempt_mode;  /* preempt mode */
    boolean_t     vi_accept_mode;   /* accept mode */
    char          vi_prot_fmri[VRRP_MAXLEN_FMRI];
                                /* SMF instances to be protected */
} vrrp_inst_t;

typedef struct vrrp_addr {
    int af;
    union {
        struct in_addr    in4;
        struct in6_addr   in6;
    } in;
} vrrp_addr_t;
```

- vi_id: VRID of this instance. Must be set when passed in.

- vi_pip: Primary IP address. If set empty (all bytes are zero) it will be derived from vi_ifname[] and vi_af.

- vi_ipnum: Must be set. 1-255.

- vi_ip: The caller must allocate the VRIP array which type is struct in_addr or struct in6_addr depending on whether vi_af is AF_INET or AF_INET6. The length of the array must be the value of vi_ipnum.

- vi_ifname[]: If set as "" it will be derived from vi_pip and vi_af. If vi_pip is empty it will be derived from vi_ip.

- vi_prio, vi_adv_intv, vi_preempt_mode, vi_accept_mode: VRRP properties, Must be set when passed in.

- vi_prot_fmri: Up to 16 SMF instances to be protected by this VRRP instance. Standard FMRI strings are used, separated by commas. When any of the SMF instances is offline, the VRRP instance will receive a SHUTDOWN event to enter the INITIALIZE state; when all the SMF instances are brought online, the VRRP instance will receive a STARTUP event to resume operation.

The vi_ifname, vi_ip, vi_pip and vi_af are used to decide which interface this instance will be running on. If vi_ifname is "", vi_pip is used. If vi_pip is empty, vi_ip is used. If the interface cannot be derived from vi_ip, an error VRRP_ENOROUTE will return indicating the interface cannot be derived from the route of the vi_ip.

The passing in vi_ip could be an existing IP address on the interface. In that case it means it is the virtual router owner. If the passing in vi_ip is found on another interface of the system, the function will return VRRP_EVRIP.

A.3.2 Destroy a VRRP instance

```
vrrp_ret_t  vrrp_destroy(const char *name);
```

Calling vrrp_destroy() will destroy the VRRP instance with the specified name. Using NULL as the parameter will destroy all VRRP instances on the system. If the required instance does not exist, the function will return VRRP_ENOINST.

This function requires the PRIV_SYS_NET_CONFIG and PRIV_NET_RAWACCESS privileges.

A.5 Send VRRP events

A.5.1 Send events to a VRRP instance

```
vrrp_ret_t  vrrp_startup(const char *name);
vrrp_ret_t  vrrp_shutdown(const char *name);
```

These functions are used to send "Startup" or "Shutdown" event to a VRRP instance. The parameter name can be set to NULL, to send event to all VRRP instances in the system. If the specified instance does not exist, the function would return VRRP_ENOINST.

This function requires the PRIV_SYS_NET_CONFIG and PRIV_NET_RAWACCESS privileges.

A.6 Access VRRP properties

A.6.1 Retrieve VRRP instance property and status information

```
vrrp_ret_t vrrp_query(const char *name, vrrp_inst_t *inst,
    vrrp_status_t *stat, vrrp_adv_info_t *adv);

typedef    uint8_t       vrrp_state_t; /* VRRP state */

typedef struct vrrp_status {
    vrrp_state_t      vs_cur_state;
    vrrp_state_t      vs_cur_substate;
    vrrp_state_t      vs_prev_state;
    vrrp_state_t      vs_prev_substate;
    struct timeval    vs_st_time;   /* timestamp of last state trans */
} vrrp_status_t;

typedef struct vrrp_adv_info {
    vrrp_addr_t       vai_addr;     /* Source IP address of the message */
    vrpri_t           vai_prio;     /* priority in adv message */
    struct timeval    vai_time;     /* timestamp of the adv message */
    struct timeval    vai_age;      /* time since the adv message */
    struct timeval    vai_intv;     /* adv interval in adv message */

    /* The following information are not available for MASTER state */
    struct timeval    vai_skew_time;    /* skew time computed for master */
    struct timeval    vai_down_intv;    /* down interval computed for mas*/
    struct timeval    vai_down_timer;   /* down timer computed for master*/
} vrrp_adv_info_t;
```

The function vrrp_query() is used to get VRRP instance's properties and status. The configuration of the instance with the specified name will be returned in the output parameter inst. The caller should allocate a vrrp_inst_t structure to be passed in. If the required instance does not exist, the function will return VRRP_ENOINST.

The output parameter stat and adv are used to return the current status of the instance and information of the last seen advertisement. The caller should allocate a vrrp_status_t structure and a vrrp_adv_info_t structure to be passed in, or pass a NULL to skip returning the status and advertisement information.

The returned status includes the current state and previous state and the timestamp when last state transition occurs. The returned vrrp_adv_info_t stores the information of the last advertisement this instance has seen. The returned adv could be set all zero, which indicates that this instance never received any advertisement.

Note that an instance in MASTER state could have this last seen advertisement information returned. In that case this last seen advertisement message could be the one received before last time this instance takes over or preempts from BACKUP state. It could also indicate a

network error where there is another router sending advertisements using the same VRID, if the timestamp of that advertisement is later than the state_trans_time of this instance.

This function doesn't require extra privileges.

A.6.3 Update a VRRP instance's properties

```
vrrp_ret_t vrrp_setprop(const char *name,
       vrpri_t priority, int delay, int preempt, int accept);

#define    VI_PROP_UNCHANGE     -1
#define    VI_PROP_PREEMPT      1
#define    VI_PROP_UNPREEMPT    0
#define    VI_PROP_ACCEPT       1
#define    VI_PROP_NOTACCEPT    0
```

The parameters priority, delay, preempt and accept can be VI_PROP_UNCHANGE which will leave the property unchanged. If the required instance does not exist the function will return VRRP_ENOINST.

This function requires the PRIV_SYS_NET_CONFIG and PRIV_NET_RAWACCESS privileges.

A.7 Look up VRRP instances

```
vrrp_ret_t vrrp_lookup(vrid_t id, const char *ifname, int af,
    int *num, char ***names);
```

This function is used to look up existing instances on the system based on VRID, interface name, and address family. Any of these three parameters could be unset, i.e. id and af could be 0, ifname could be NULL. The functions would return all instances matching the given attributes.

The number of found instances are returned in `num`, and the names of all found instances are returned in the `names` array. If the returned `num` is not 0, the caller should free the array.

This function doesn't require extra privileges.

Appendix B. vrrpadm CLI Specifications

1. Summary

In the VRRP implementation there will be two programs: (1) vrrpd, the daemon, and (2) vrrpadm, the admin tool to configure the daemon. A SMF service

svc:/network/vrrp:default

will also be introduced.

When the VRRP service is enabled, the vrrpd daemon is started with the default configuration file, as "vrrpd -f /etc/vrrpd.conf", and the VRRP instances defined in this configuration file will be started. When the service is running, the "vrrpadm" command can be used to create, delete, update VRRP instances.

2. The Daemon

The VRRP daemon, or vrrpd, reads the content of the configuration file. The location of the configuration file is a configurable property of the svc:/network/vrrp:default SMF service. By default it is at /etc/vrrpd.conf.

The format of the configuraton file will be detailed in section B.5. Note, the user is not recommended to edit the configuration file manually.

3. The vrrpadm tool

The vrrpadm tool is used to communicate with the daemon to create new VRRP instances, delete existing ones, modify the properties of existing ones, etc.

The vrrpadm command looks like:

```
vrrpadm <subcmd> <options> <vname>
```

The sub-commands are:

`create`: to create and start running.

`startup`: to send a "startup" event to the VRRP instance. The VRRP instance will leave INIT state on receiving it.

`shutdown`: to send a "shutdown" event to the VRRP instance. The VRRP instance will enter INIT state on receiving it.

`delete`: to delete from the system.

`modify`: to modify run-time properties.

`show`: to show properties.

Each command ends with the VRRP instance name, which uniquely identifies the VRRP instance to operate on.

Most of these commands (create, startup, shutdown, delete, modify) require the PRIV_SYS_NET_CONFIG and PRIV_NET_RAWACCESS privileges to operate; the "show" command doesn't require extra privileges.

3.1 Creation

To create a new VRRP instance:

```
vrrpadm create
    -v <vrid> -i <intf> [-j <primary_addr>] -A <ipaddr_list>
    [-p <priority>] [-d <adv_interval>] [-o <flags>]
    [-P <protected_service>[,...]]
    <vinst_name>
```

Note, the user must specify a name for the instance to be created. The name must be a string consisting of letters, digits and underscores.

`-p <priority>`:
The priority of this host. Default value: 1.

`-d <adv_interval>`:
The advertisement interval, in seconds. Default value: 1.

`-o <flags>`:
The preempt and accept modes, delimited by a comma. Values can be: `preempt`, `un_preempt`, `accept`, `not_accept`. By default both will be set to true.
Example: `-o preempt,not_accept`.

`-v <vrid>`:
the VRID. Value range: 1-255.

`-i <intf>`:
The interface on which this virtual router is hosted.

`[-j <primary_addr>]`:
The primary address through which the VRRP advertisements are sent and received. This address should be configured on `<intf>` before vrrpd is launched.

`-A <associate_addrs>`:
The ip address(es) associated with the virtual router, delimited with a comma. Both IPv4 and IPv6 addresses are supported.

`[-P <protected_service>[,...]]`:
The SMF services to be protected by this VRRP instance, delimited with commas.

Example:

```
vrrpadm create -v 13 -i nge1 -j 11.12.13.90 -p 200 -A
    192.168.10.10,192.168.10.11 -P svc:/network/http:apache2 vinst13
```

3.2 Startup

To leave the INITIALIZE state (and eventually enter BACKUP or MASTER state):

```
vrrpadm startup <vinst_name>
```

A startup command sent to a VRRP instance whose protected service is disabled/offline will fail.

3.3 Shutdown

To enter the INITIALIZE state from MASTER or BACKUP state:

```
vrrpadm shutdown <vinst_name>
```

3.4 Deletion

To delete a VRRP instance from the system:

```
vrrpadm delete <vinst_name>
```

3.5 Modification

To modify some run-time properties of an existing VRRP instance:

```
vrrpadm modify
   [-p <priority>] [-d <adv_interval>] [-o <flags>]
   <vinst_name>
```

Note, only the priority, advertisement interval, the preempt mode and the accept mode can be modified. Other parameters such as `<intf>` and `<ipaddr_list>` are considered "fatal"; if they're to be changed, it is better to stop the current VRRP instance and start a new one to replace it.

3.6 Show

To show VRRP configuration and state:

```
vrrpadm show [-p] [-o field[,...]] [<vinst_name>]
```

If the VRRP instance name is not specified, it will show all VRRP instances in the system.

“-o” specifies a list of fields to display, separated with commas. “-p” displays in a machine parsable format; the fields are separated by colons. Note “-p” must use together with “-o”. By default it displays all fields in human readable format.

(1) The verbose, human readable display of a VRRP instance:

```
$ vrrpadm show vinst4
INSTANCE vinst4
     vrid = 4
     priority = 100
     adv_intval = 2
     preempt_mode = true
     accept_mode = false
     interface = bge2
     primary_ipaddr = 10.100.77.77
     assoc_ipaddrs = 10.100.77.88,10.100.77.99
     protected_service = svc:/network/http:apache2
     prev_state = INIT
     time_last_state_trans = 2008.03.12,16:34:08.323 (10.323 secs ago)
     state = BACK
         master_priority = 255
         master_src_addr = 10.100.77.78
         master_adv_intval = 1.00
         master_adv_last = 2008.03.12,16:44:08.323 (0.323 seconds ago)
         skew_time = 0.391
         master_down_intval = 3.391
         master_down_timer = 3.068
```

Note the content from "state = xxxx" is different for different states. For the MASTER state, it looks like:

```
     state = MAST
         peer_priority = 255
         peer_src_addr = 10.100.77.78
         peer_adv_intval = 1.00
         peer_adv_last = 2008.03.12,16:44:08.323 (0.323 seconds ago)
```

Note the names of the fields are "peer_xxx" instead of "backup_xxx", and some fields (skew_time, master_down_intval, master_down_timer) don't apply to this state and are thus not displayed.

If the state is INIT, it doesn't has specific data except for the sub-state:

```
  state = INIT
    sub_state = shutdown
```

Or:

```
  state = INIT
    sub_state = pending
```

(2) You can select several fields to display in the human readable format.

```
$ vrrpadm show -o vrid,interface,primary_ipaddr,state vinst4
INSTANCE vinst4
     vrid = 4
     interface = bge2
```

```
    primary_ipaddr = 10.100.77.77
    state = BACK
```

(3) Machine parsable output is designed for easy parse for scripts. The fields are separated by colons. “-p” must be used together with “-o”.

```
$ vrrpadm show -p -o name,vrid,interface,primary_ipaddr,state
vinst4:4:bge2:10.100.77.77:BACK
vinst13:13:nge1:192.168.3.3:MAST
```

4. The configuration file

Using the command line options, we can only start one VRRP instance at a time -- the options will otherwise be too complex to parse. On the contrast, using a configuration file, we can specify multiple VRRP instances at a time.

The configuration file defines one or more VRRP instances, each of which is in a block surrounded by curly brackets.

A VRRP instance is configured as:

```
vrrp_instance {
    name              <vinst_name>
    active            true | false
    priority          <priority>
    adv_intval        <adv_interval>
    preemp_mode       enabled | disabled
    accept_mode       enabled | disabled

    vrid              <vrid>
    interface         <intf>
    primary_ipaddr    <primary_ipaddr>
    assoc_ipaddrs     <associated_ipaddrs>

    protected         <protected_service>

    ...
}
```

Note, if "active" is set to true, the instance will start running normally after being created. If "active" is set to false, the instance will stay in the INIT state on creation, as if a "shutdown" event is received.

Example: a configuration file that defines two VRRP instances.

```
### Beginning of example configuration file

vrrp_instance {
    name              vinst3
    active            false
    priority          200
    adv_intval        2
    preempt_mode      disabled
```

```
    accept_mode       disabled

    vrid              13
    interface         nge1
    primary_ipaddr    192.168.10.8
    assoc_ipaddrs     192.168.10.10,192.168.10.11

    protected         svc:/network/http:apache2

}

vrrp_instance {
    name              vinst2
    active            true
    priority          225
    adv_intval        1
    preempt_mode      enabled
    accept_mode       enabled

    vrid              20
    primary_ipaddr    192.168.10.200
    assoc_ipaddrs     192.168.10.98
}

### End of example configuration file
```

Change Log

0.9.4. minor changes to sysevent mechanism (4.4)
- added more error types (3.4, A.1)
- removed vrrp_name_t and related APIs (3.4, A)
- added sub-state for INITIALIZE state (4.2, 4.3, 5.5)
- added more discussion for SMF interaction (5.7)

0.9.3. Added introduction for service protection
- changed vrrp_create() and added "-P fmri" option to support protecting fmri (A.3.1)
- added description for service protection and its implementation (3.1, 4.2, 5.7)
- revised threading description (5.8)
- removed all group discussions
- added sysevent discussions
- used "-o fields" method for machine parsable format show
- refractor the CLI/API discussions and appendix

0.9.2. Removed SMF discussion.

0.9.1. The first stable beta version to put onto OpenSolaris.org website.