VRRP
(Virtual Router Redundancy Protocol)
Design Document

Revision 0.9.1, Nov. 07, 2008

vrrp-core@sun.com

Table of content

1. Introduction

This project is to implement the VRRP (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.

The VRRP standard is specially designed for routers. To make it usable in a wider range, we extend the VRRP protocol in the following two aspects:

- Introduce VRRP group which binds up multiple VRRP instances to perform fail over as a whole and thus provides the capability of protecting IP addresses which are configured on multiple NICs.
- Provide an option that VRRP instances can be configured as non-router type to skip sending Neighbor Discovery Router Advertisements as required by the standard.

With the extension to the standard as above, the VRRP service can be used for high availability on other network appliances such as load balancers and firewalls.

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 and group

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 a 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.

The concept of "VRRP group" on a host is introduced as an extension to the standard. A VRRP group contains several VRRP instances, and all these instances always have the same state. The group changes its state only when all of its instances meet the criteria of state transition. Thus a VRRP group binds up multiple VRRP instances and performs fail over as a whole. See

section 3.4 for details of VRRP group.

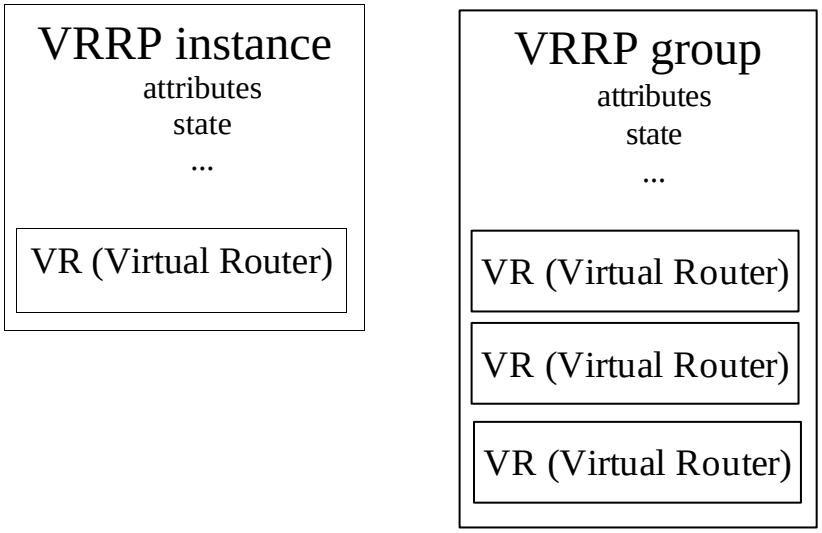

Figure 3.1

To distinguish between VRRP single instance and VRRP group, the term "VRRP instance" is used only to refer to a VRRP config holding one single instance, while the term "VRRP group" is used to refer to a VRRP config holding multiple instances in its virtual router. A term "Virtual Router" or "VR" is used instead to refer to the instance contained in a VRRP group. A VRRP group contains more than one Virtual Routers, while a VRRP instance contains only one Virtual Router.

A VRRP instance consists of:

- Instance name (identifier, system global unique)
- Attributes: priority, advertise interval, preempt mode, accept mode
- One Virtual Router (VR)
- State information and statistics

A VRRP group consists of:

- Group name (identifier, system global unique)
- Attributes: priority, advertise interval, preempt mode, accept mode
- More than one Virtual Routers (VR)
- State information and statistics

A Virtual Router (VR) consists of:

- VRID, interface name, address family
- Primary IP used as the source of the advertisement
- Associated IPs to be protected

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 program to exploit
- A repositary vrrpd.conf for persistent configuration

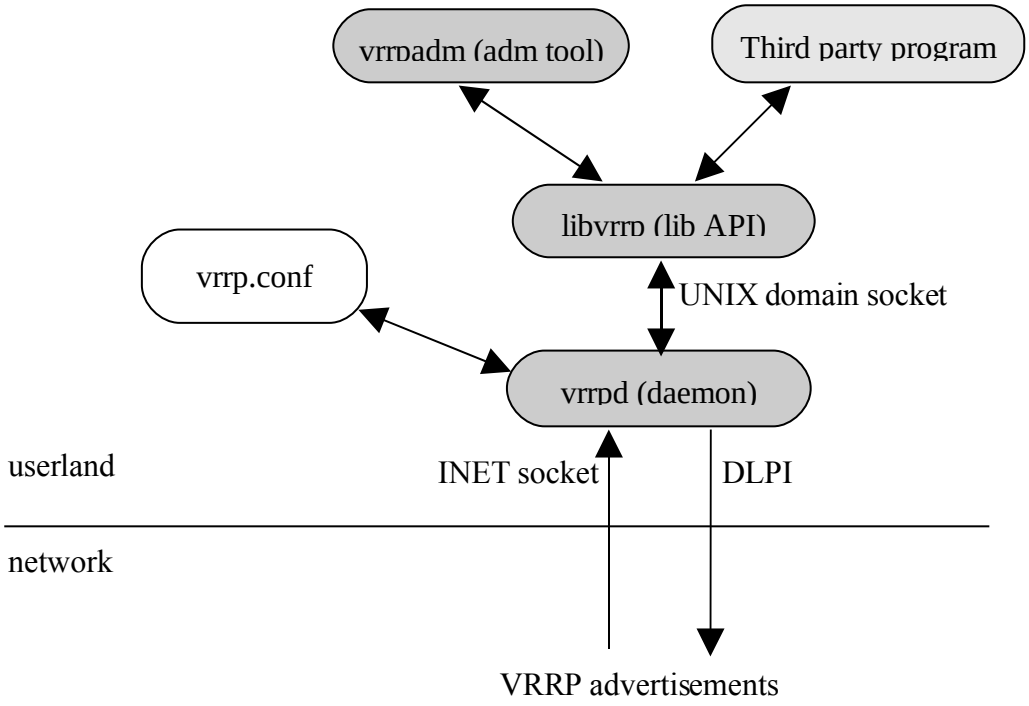

Figure 3.2

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

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

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 Basic operations

Most operations apply to both an instance or a group.

The "vrrpadm create" command is used to create a VRRP instance or a group. To do this using libvrrp APIs, use vrrp_create_inst() or vrrp_create_grp(), respectively. Note, when creating an instance, the inst_name must be provided; when creating a group, the grp_name must be provided, and the instances within this group don't need names.

After an instance or a group is created, its name is used in the operations that follows.

The user then can use "vrrpadm modify" or vrrp_set_inst_prop() or vrrp_set_grp_prop() to modify properties of an instance or a group. 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.

For a running VRRP instance/group, "vrrpadm startup" or vrrp_startup_inst() or vrrp_startup_grp() can be used to send a "STARTUP" event to it. On receiving this event, the VRRP instance/group, if in the INITIALIZE state, will leave the INITIALIZE state and enter either BACKUP or MASTER state finally. Similarly, "vrrpadm shutdown" or vrrp_shutdown_inst() or vrrp_shutdown_gro() can be used to send a "SHUTDOWN" event; on receiving it, the VRRP instance/group, if in BACKUP or MASTER state, will enter the INITIALIZE state, which is actually a "paused" state.

Finally, "vrrpadm remove" or vrrp_destroy_inst() or vrrp_destroy_grp() are used to remove a VRRP instance/group from the system.

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/group is shut down, the corresponding "active" is set to true, and next time when vrrpd is started, the "active" flag will cause the instance/group to stay in the INITIALIZE state.

There's also the privilege issues - not all users are allowed to do all operations. See section 9 for details.

3.4 VRRP group discussion

The VRRP standard is designed for routers, defining a set of virtual router

associated IP addresses on a single interface, which usually act as the default route for internal machines. The VRRP protocol is incapable of protecting IP addresses on multiple interfaces. Thus it cannot work for some other network appliances such as load balancers. In this project we extend the standard protocol and introduce VRRP group, which binds up multiple VRRP instances and performs fail-over as a whole.

The difference from running two independent VRRP instances is that VRRP group guarantees that all instances in the group always have the same state. There are cases where only one instance doing fail over among multiple instances doesn't work for the whole network setup. There might be need that all instances either do fail over as a whole, or rather stay unchanged.

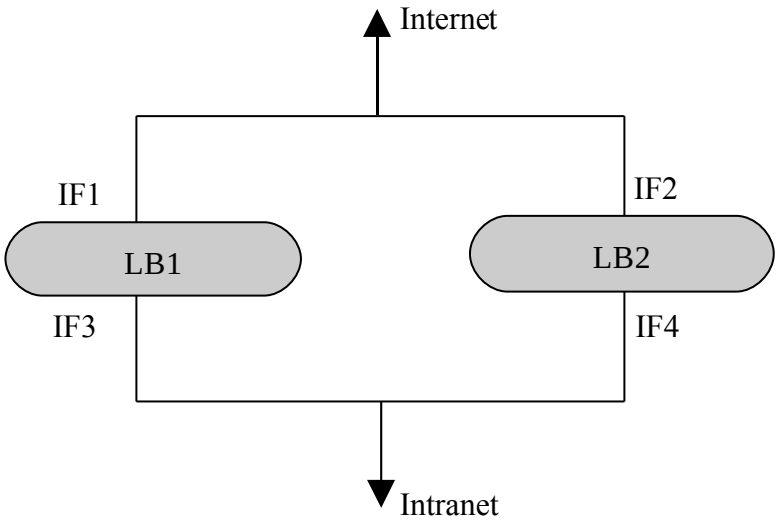

Figure 3.3

For example, consider a pair of load balancers LB1 and LB2 as graphed in figure 3.3, where both outside IP and the inside IP need to be protected by VRRP. The outside IP is a public IP address for serving services to the Internet, while the inside IP is the destination of the default route of a set of back end servers. If VRRP is configured as two independent instances applied on inside interfaces (IF1 and IF2) and outside interfaces (IF3 and IF4) respectively, when the inside link IF3 somehow fails, the inside IP will fail over to the backup on IF4, while the outside IP will stay on the original master LB1. That breaks the network setup and no traffic will be able to send in and out.

By grouping the inside and outside instances, the uneven fail over case can be avoided. But a decision needs to be made when one or some of the instances fail. Should the group do fail-over? One choice is to only trigger fail-over until

all instances fail, another choice is do mandatory fail over for all instances whenever there is failed instance.

It is a trade-off choosing either one. The first solution is unable to protect the network from the above failure case. The second solution can work for cases that part of links on master fail, as described in the example. But there is possibility that it is the backup's inside link (IF4) who fails and hence can't hear advertisement from the master. In this case although LB1 is actually working fine, due to the false-positive from IF4, both protected IPs would be taken over by LB2 and consequently break the network. The advantage of the first solution is it can avoid such a false-positive.

Another issue of the second solution is lack of support from the VRRP protocol. It is the backup who detects absence of advertisement and performs fail over. But the backup has lower priority than the master. So in the above example, if the backup wants to do mandatory fail over for both instances, it needs a way to preempt the power from the master on the outside instance, such as changing its own priority to a higher one than that of the master. However the protocol doesn't provide such a scheme.

Considering the issues of the second solution, we only provide the first solution in this implementation. To solve the failure cases that only part of the links fail in a group, users should seek assistances from other components such as a health checking service to detect these kind of failures and consequently trigger mandatory fail-over through the management interface of the VRRP service, for example, send shutdown event to VRRP on the master.

3.5 SMF services discussion

3.5.1 Summary

The VRRP protocol is designed to protect routers, and the direct motivation of the project is to protect the ILB service. People may also want to protect other popular services such as http, ftp, etc. Services such as forwarding and (integrated) load balancing are provided by the kernel (or, from the admin's point of view, by the whole system), while http and ftp are provided by applications. Applications must use bind() to bind to the publicly known server address before they can provide service, but kernel-provided services don't have this requirement.

Before VRRP is running, the IP address to be protected may not appear on the system, so the applications have no way to bind to it. To solve this, a VRRP instance must be created to set up the IP address before the application is started.

Another difference between “kernel-provided” services and application services is that for the former "service failure" means the whole host failure, and the detection of the service failure is achieved by the peer not receiving the advertisements. For application services, "service failure" has to rely on some external health check mechanism to detect, and then explicitly bringdown the master instance to trigger fail over. Using the SMF dependency mechanism seems to be straightforward to perform this task. If VRRP instance is configured as a SMF instance and has dependency on the protected service, then the fail-over could be triggered by the SMF mechanism. But as stated as above, the VRRP instance must set up the IP address before the application is started, this becomes a mutal dependency error so it could not work in this way.

In a summary, for applciation services, before it is started, VRRP must be running to set up the IP address; after the service fails, VRRP must be stopped to trigger fail-over. A solution is proposed to solve this problem. The SMF attributes of the services to be protected are modified to perform this task.

3.5.2. Interact with SMF services

The VRRP introduces the SMF service as svc:/network/vrrp:default and the configuration file vrrpd.conf stores the configurations for all permanent VRRP instances/groups, to protect kernel services such as forwarding and load balancing. This is the "standard" use case.

For application services, as stated in 3.5.1, VRRP instances must be started before the service is started, and should be stopped after the service failure. This is achieved by modifying the start and stop method of the protected SMF service. Take http:apache2 as an example. In the start method, before starting the apache program, the VRRP instance is created using the "vrrpadm create" command to set up the associated IP addresses so that apache can successfully bind to it; in the stop method, after killing the apache program, the VRRP instance is deleted using the "vrrpadm remove" command so that the peer VRRP will no longer receive VRRP advertisements. If http:apache2 is killed by accident or disabled intentionally using "svcadm disable", the stop method is called and then VRRP instance is destroyed. (When killed by accident, SMF

will probably call the start method to bring it online again. The time gap between VRRP instance destroying and recreation should be short enough so that the peer won't become master.)

The VRRP instances associated with application services should be "temporary", i.e., their configurations shouldn't be stored in the configuration file, otherwise they'll be created by the vrrp:default SMF service after the system boots, separately from the services they are to protect. (Note, technically, making such VRRP instances persistent also work - the start method will fail to create it with "object already exists", but the stop method can successfully remove them. But logically such VRRP instances should be bundled with the protected services.)

If multiple services use the same IP address, they can be protect by a single VRRP instance. The same "vrrpadm create" method should be added to the start method of each of these services. Only one of them will successfully create the instance, and the others will fail with "object already exists". Similarly, the stop method of all services have the same "vrrpadm destroy" command, thus the first failure of these services will cause the fail-over. This souds like a "require_all" dependency in SMF terms, although here the protected SMF service doesn't have a SMF dependency with the VRRP instance.

The user doesn’t have to manually modify the start and stop method of existing services. Instead, we provide a tool, vrrpize, to do this. If an administrator wants to put a certain service under VRRP protection, he can use vrrpize to make the modifications, either overwrite the existing service or create a new one.

Thus, the VRRP instances in the system fall into two categories:

(1) Permanent, used to provide kernel services. Their configurations are stored in the config file. They are created by the vrrp:default SMF services after system boots.

(2) Temporary, use to protect application-provided services. They are created and deleted by the start and stop method of the protected service, respectively.

4. State transition and Failover

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

4.1 manual startup or shutdown

In our model, 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 or group that is in the INITIALIZE state using either "vrrpadm startup" sub-command or the vrrp_startup_inst/grp() APIs.

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

There are several reasons that there is need to have an INITIALIZE state in a practical VRRP implementation: (1) the RFC clearly defines this state and behaviors associated with this state; (2) This state can be seen as a "paused" state and is useful when starting VRRP instances on multiple machines: the protocol may not function well if a VRRP instance on one machine is working while its peer on another machine hasn't started. With INITIALIZE state, we can first create VRRP instances on different machines one by one, putting all VRRP instances in the INITIALIZE state; when all VRRP instances on all machines are created, we send a STARTUP command to them so they can start working at almost the same time.

We also extend the concept of state transfer to VRRP groups. When a STARTUP or a SHUTDOWN event is sent explicitly to a group, all VRRP instances in this group will do the state transfer.

4.2 fail-over of a group

The more important case is that when the user is not involved: the group will make the state transfer as a whole only when all its VRRP instances need to do that. Thus, for example, a VRRP group is in BACKUP state, i.e., all its member VRRP instances are in BACKUP state. If one (or more, but not all) VRRP instances detects that its master fails, it won't transfer to the MASTER

state, unless all the rest of VRRP instances in the group detects so. The transfer will happen only when all VRRP instances in the group detects the failure of their corresponding masters.

4.3 User-defined Actions Associated with State Transitions

A mechanism is provided so that when a state transition occurs, a user-defined action can be taken.

If an executable (binary or script), `/etc/vrrpd/sthook.<name>`, exists, where `<name>` is the unique name of a VRRP instance or group, the vrrpd daemon will automatically run the executable when the VRRP instance or group identified by `<name>` has a state transition. The state transition can be one of the followings: `INIT_TO_BACK`, `INIT_TO_MAST`, `BACK_TO_MAST`, `MAST_TO_BACK`, `MAST_TO_INIT`, `BACK_TO_INIT`.

By default the system doesn't provide default executables. The user is responsible for putting files under `/etc/vrrpd/`. These executable files are expected to owned by the root and have 755 mode.

The state transition name as listed above will be passed as the argument to run the executable. The executable runs with the same privileges and environment as its caller, the vrrpd daemon, except that the stdin, stdout and stderr are redirected to /dev/null. Note this means the executable will run as root, so the user should clearly know what he's doing. See Section 9.2 for related security discussions.

5. libvrrp library

This project will provide a set of APIs in libvrrp.so; each API has a corresponding vrrpadm sub-command. Third party softwares can use these APIs to write their own tools to configure VRRP.

A VRRP instance is uniquely identified by an instance name. Likewise, a VRRP group is identified by a group name. Most APIs that handles a VRRP instance or a VRRP group have a name as the input parameter.

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

An enum vrrp_ret_t is used to store the return status of each API functions.

```
typedef enum {
        VRRP_SUCCESS = 0,
        VRRP_EVREXIST,      /* vrrp virtual router already exists */
        VRRP_EINSTEXIST,    /* vrrp instance name already exists */
        VRRP_EGRPEXIST,     /* vrrp group name already exists */
        VRRP_EINVAL,        /* invalid parameter */
        VRRP_ENOROUTE,      /* cannot determine interface by vr_pip */
        VRRP_EVRIP,         /* invalid virtual router ip */
        VRRP_ENOINST,       /* vrrp instance does not exist */
        VRRP_ENOGRP,        /* vrrp group does not exist */
        VRRP_ENOPERM,       /* insufficient privilege */
        VRRP_ESYS           /* system error */
} vrrp_ret_t;
```

5.1 Create and destroy a VRRP instance or a VRRP group

The following two functions are used to create a VRRP instance and a VRRP group.

```
vrrp_ret_t vrrp_create_inst(vrrp_inst_t *inst);
vrrp_ret_t vrrp_create_grp(vrrp_grp_t *grp, vrrp_vr_t *vr);

typedef struct vrrp_inst {
      vinst_name_t   vi_name;    /* vrrp instance name */
      vrrp_attr_t    vi_va;      /* vrrp attributes */
      vrrp_vr_t      vi_vr;      /* virtual router attribute set */
      boolean_t      vi_active;  /* Start up? */
} vrrp_inst_t;

typedef struct vrrp_grp {
      vgrp_name_t    vg_name;   /* vrrp group name */
      vrrp_attr_t    vg_va;     /* vrrp attributes */
      uint8_t        vg_vrnum;  /* number of virtual routers this group has. 2-255 */
      boolean_t      vg_active; /* Start up? */
} vrrp_grp_t;


typedef struct vrrp_vr {          /* virtual router */
      vrid_t         vr_id;       /* VRID of this vr */
      vrrp_addr_t    vr_pip;      /* primary IP */
      uint8_t        vr_ipnum;    /* number of virtual router IP, 1-255 */
      void           *vr_ip;      /* pointer to the virtual IP array */
      char           vr_ifname[LIFNAMSIZ];
} vrrp_vr_t;
```

```
typedef struct vrrp_attr {
      vrpri_t       va_pri;             /* priority */
      int           va_delay;           /* advertisement interval */
      boolean_t  va_preempt_mode;       /* preempt mode */
      boolean_t  va_accept_mode;   /* accept mode */
} vrrp_attr_t;
```

The vrrp_vr structure is used to store a Virtual Router, which contains the attributes for a VRRP router per VRID. A VRRP instance structure contains one vrrp_vr structure, while a VRRP group has a vrrp_vr array linked to it. See appendix A for detailed information on data structures.

The following two functions are used to destroy existing VRRP instances or groups.

```
vrrp_ret_t      vrrp_destroy_inst(vinst_name_t name);
vrrp_ret_t      vrrp_destroy_grp(vgrp_name_t name);
```

5.2 Send events to a VRRP instance or a VRRP group

```
vrrp_ret_t      vrrp_startup_inst(vinst_name_t name);
vrrp_ret_t      vrrp_shutdown_inst(vinst_name_t name);

vrrp_ret_t      vrrp_startup_grp(vgrp_name_t name);
vrrp_ret_t      vrrp_shutdown_grp(vgrp_name_t name);
```

These functions are used to send "STARTUP" or "SHUTDOWN" event to a VRRP instance or a VRRP group. See section 3.4 for details of events and state transition.

5.3 Look up instances or groups

```
vrrp_ret_t      vrrp_lookup_inst(vrid_t id, const char *ifname, int af,
            int *num, vinst_name_t *names);
vrrp_ret_t      vrrp_lookup_grp(vrid_t id, const char *ifname, int af,
            int *num, vgrp_name_t *names);
```

These two functions are used to look up existing instances or groups on the system based on VRID, interface name, and addresses 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 or groups matching the given attributes.

The number of found instances or groups are returned in the output parameter num, and the names of all found instances or groups are returned as a "char [][VNSIZ]" array through the ourput parameter names. If the returned num is not 0 and names is not NULL, the caller should free the array.

5.4 Update a VRRP instance or a VRRP group's properties

To modify the properties of a VRRP instance or a VRRP group, the following two functions are used.

```
vrrp_ret_t      vrrp_set_inst_prop(vinst_name_t name,
                vrpri_t priority, int delay, int preempt, int accept);
vrrp_ret_t      vrrp_set_grp_prop(vgrp_name_t name,
                vrpri_t priority, int delay, int preempt, int accept);
```

Only the priority, the delay, and the preempt mode can be modified for an existing instance or group. To change the other properties one must destroy and recreate the instance or the group.

5.5 Access VRRP properties

```
vrrp_ret_t      vrrp_get_inst(vinst_name_t name, vrrp_inst_t *inst,
                 vrrp_status_t *stat, vrrp_adv_info_t *adv);
vrrp_ret_t      vrrp_get_grp(vgrp_name_t name, vrrp_grp_t *grp,
                   vrrp_status_t *stat, vrrp_vr_t **vr, vrrp_adv_info_t **adv);

typedef uint8_t         vrrp_state_t;               /* VRRP state */

typedef struct vrrp_status {
      vrrp_state_t    vs_current_state;
      vrrp_state_t    vs_previous_state;
      struct timeval  vs_state_trans_time;    /* timestamp of last state transition */
} vrrp_status_t;

typedef struct vrrp_adv_info {
      vrrp_addr_t     vai_addr;         /* Source IP address of the message */
      vrpri_t         vai_priority;     /* 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_interval;   /* adv interval in adv message */

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

These two functions are used to get properties and status of VRRP instances and VRRP groups respectively. The configuration of the instance with the specified name will be returned in the output parameter inst and grp respectively, and the status information will be returned in the output parameter stat. The caller should allocate a vrrp_inst_t structure or a vrrp_grp_t structure as well as a vrrp_status_t structure to be passed in. To retrieve all properties from VRRP

group, the output parameter vr is used to return an array of the vrrp_vr_t structures containing the properties of all the virtual routers.

The output parameter adv is used to return the information of the last seen advertisement. The caller should allocate a vrrp_adv_info_t structure for vrrp_get_inst() to be passed in. For vrrp_get_grp(), the function will allocate a vrrp_adv_info_t array to return and the caller should free the returned array.

6. Configuration file

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

The configuration file uses a nested block-structured syntax similar to what is used for gated.conf. The whole configuration file looks like:

```
vrrp_group {
      group_name      vrgp3;
        :
      vrrp_instance {
              vrid    12;
              interface e1000g1;
```

```
                :
        }
        vrrp_instance {
                vrid    13;
                interface nge1;
                :
        }
}

vrrp_instance {
   inst_name   inst2;
   vrid  14;
   interface nge2;
   :
}
```

For more details about configuration file see appendix B.

7. Implementation details

7.1 Communication between vrrpadm and vrrpd

When started, vrrpd creates the VRRP instances and/or groups 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.

7.2 Sending and receiving VRRP advertisements

VRRP instances/groups in master send periodic advertisements as required by the RFC specification. VRRP instances/groups in 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 (because Solaris doesn't have a type of socket that has access to raw datalink, as what PF_PACKET does in Linux). 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.

7.3 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_vr_t.vr_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.

7.4 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.

7.5 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

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

7.6 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.

7.7 Timers and threading

Each VRRP instance or VRRP group in master state has a advertisement timer that fires to trigger sending of advertisement. Each VRRP instance or VRRP group in backup state has a master down timer that fires when advertisement has not been heard for specified period of time, to trigger state transition. To minimize the complexity, the 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 multi-process is not needed for vrrpd implementation.

8. 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 or VRRP groups could be configured on aggregated interfaces and work properly.

9. Security considerations

9.1 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 vrrpadm show commands and vrrp_get_*() vrrp_lookup_*() routines. Other commands and APIs require PRIV_SYS_NET_CONFIG and PRIV_NET_RAWACCESS privileges.

9.2 About User-defined Actions Assicoated with State Transitions

As discussed in Section 4.3, the vrrpd daemon runs as root, and all executables associated with state transitions also runs as root.

An alternative is to introduce a new user “vrrp”, and grant processes run as “vrrp” user limited privileges -- PRIV_SYS_NET_CONFIG and PRIV_NET_RAWACCESS, as discussed in Section 9.1. The vrrpd daemon runs as user “vrrp”, and thus all executables associated with state transitions as runs as “vrrp”, with limited privileges. The drawback is that, these executables

may need higher privileges to do certain jobs. For example, when VRRP is used to protect other services, it's very likely to manipulate other programs or SMF services. This may need more privileges.

So using “root” is simple in design, but this requires that the user is very clear about what he needs. The dhcpagent(1M) program also uses root similarly.

9.3 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_EVREXIST,    /* vrrp virtual router already exists */
    VRRP_EINSTEXIST,  /* vrrp instance name already exists */
    VRRP_EGRPEXIST,   /* vrrp group name already exists */
    VRRP_EINVAL,      /* invalid parameter */
    VRRP_ENOROUTE,    /* cannot determine interface by vr_pip */
    VRRP_EVRIP,       /* invalid virtual router ip */
    VRRP_ENOINST,     /* vrrp instance does not exist */
    VRRP_ENOGRP,      /* vrrp group 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. Likewise, a VRRP group is identified by a group name. Most APIs that handles a VRRP instance or a VRRP group have a name as the input parameter.

Two data types are defined for their names:

```
typedef __vinst_name *vinst_name_t;    /* VRRP inst name */
typedef __vgrp_name  *vgrp_name_t; /* VRRP group name */
```

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

Functions are provided to convert between these names and strings:

```
boolean_t  vrrp_str_to_instname(const char *str, vinst_name_t name);
boolean_t  vrrp_instname_to_str(vinst_name_t name, char *str);

boolean_t  vrrp_str_to_grpname(const char *str, vgrp_name_t name);
boolean_t  vrrp_grpname_to_str(vgrp_name_t name, char *str);
```

Each of them returns `B_TRUE` or `B_FALSE` to indicate whether the conversion is successful. Errno is set to `VRRP_EINVAL` if the conversion fails due to an invalid name.

A.3 VRRP instance

A.3.1 Create a VRRP instance

```
vrrp_ret_t  vrrp_create_inst(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_inst()`.

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 `INITIATE` state.

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

#define    VRRP_STATE_INIT   1
#define    VRRP_STATE_MAST   2
#define    VRRP_STATE_BACK   3

#define    VNSIZ  16

typedef struct vrrp_inst {
    vinst_name_t  vi_name;   /* vrrp instance name */
    vrrp_attr_t   vi_va;     /* vrrp attributes */
    vrrp_vr_t     vi_vr;     /* virtual router attribute set */
    boolean_t     vi_active; /* Start up? */
} vrrp_inst_t;

typedef struct vrrp_vr {    /* virtual router */
    vrid_t vr_id;           /* VRID of this vr */
    int vr_af;              /* IP family */
    vrrp_addr_t   vr_pip;   /* primary IP */
    uint8_t    vr_ipnum;    /* number of virtual router IP, 1-255 */
    void   *vr_ip;          /* pointer to the virtual IP array */
    char   vr_ifname[LIFNAMSIZ];
} vrrp_vr_t;

typedef struct vrrp_attr {
    vrpri_t       va_pri;       /* priority */
    int           va_delay;     /* advertisement interval */
    boolean_t     va_preempt_mode;  /* preempt mode */
    boolean_t     va_accept_mode;   /* accept mode */
} vrrp_attr_t;

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

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

- `vr_pip`: Primary IP address. If set empty (all bytes are zero) it will be derived from `vr_ifname[]` and `vr_af`.

- `vr_ipnum`: Must be set. 1-255.

- `vr_ip`: The caller must allocate the VRIP array which type is struct in_addr or struct in6_addr depending on whether vr_af is AF_INET or AF_INET6. The length of the array

must be the value of vr_ipnum.

- `vr_ifname[]`: If set as "" it will be derived from `vr_pip` and `vr_af`. If `vr_pip` is empty it will be derived from `vr_ip`.

- `va_pri, va_delay, va_preempt_mode, va_accept_mode`: VRRP properties, Must be set when passed in.

The `vr_ifname, vr_ip, vr_pip` and `vr_af` are used to decide which interface this instance will be running on. If `vr_ifname` is "", `vr_pip` is used. If `vr_pip` is empty, `vr_ip` is used. If the interface cannot be derived from `vr_ip`, an error `VRRP_ENOROUTE` will return indicating the interface cannot be derived from the route of the `vr_ip`.

The passing in `vr_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 vr_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_inst(vinst_name_t name);
```

Calling `vrrp_destroy_inst()` will destroy the VRRP instance with corresponding 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.4 VRRP group

A.4.1 Create a VRRP group

```
vrrp_ret_t  vrrp_create_grp(vrrp_grp_t *grp, vrrp_vr_t *vr);
```

To create a VRRP group, one should use a vrrp_grp_t structure in conjunction with an array of vrrp_vr_t to call vrrp_create_grp(). If the property vg_active is B_TRUE, the created VRRP group will get a "Startup" event upon creation. If vg_active is B_FALSE, the created VRRP group will be in INITIATE state.

This function requires the PRIV_SYS_NET_CONFIG and PRIV_NET_RAWACCESS privileges.

```
typedef struct vrrp_grp {
    vgrp_name_t   vg_name;   /* vrrp group name */
    vrrp_attr_t   vg_va;     /* vrrp attributes */
    uint8_t       vg_vrnum;  /* number of virtual routers */
```

```
                            /* this group has. 2-255 */
    boolean_t     vg_active; /* Start up? */
} vrrp_grp_t;
```

The vrrp_grp_t structure should contain necessary information for setting VRRP group properties. See structure comments for property description.

Group names have own namespace on each system, just like instance names do. Using an existing group name to create a group would cause the function to return VRRP_EGRPEXIST.

The value of vg_vrnum indicates the array length of parameter vr. The caller should allocate a vrrp_vr_t array to be passed in.

For each element of the vrrp_vr_t array passed in, caller should follow the rules in 1.1 to set the properties. Note vr_af of different vrrp_vr_t can be different.

A.4.2 Destroy a VRRP group

```
vrrp_ret_t  vrrp_destroy_grp(vgrp_name_t name);
```

Calling vrrp_destroy_grp() will destroy the VRRP group with the passing in group name. Using NULL as the parameter will destroy all VRRP groups on the system. If the required group does not exist, the function will return VRRP_ENOGRP.

This function requires the PRIV_SYS_NET_CONFIG and PRIV_NET_RAWACCESS privileges.

A.5 Send VRRP events

A.5.1 Send events to VRRP instance/group

```
vrrp_ret_t  vrrp_startup_inst(vinst_name_t name);
vrrp_ret_t  vrrp_shutdown_inst(vinst_name_t name);

vrrp_ret_t  vrrp_startup_grp(vgrp_name_t name);
vrrp_ret_t  vrrp_shutdown_grp(vgrp_name_t name);
```

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

These functions require 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_get_inst(vinst_name_t 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_current_state;
    vrrp_state_t      vs_previous_state;
    struct timeval    vs_state_trans_time; /* timestamp of last state */
                                           /* transition */
} vrrp_status_t;

typedef struct vrrp_adv_info {
    vrrp_addr_t       vai_addr;     /* Source IP address of the message */
    vrpri_t           vai_priority; /* 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_interval; /* 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_interval;   /* down interval computed */
                                           /* for master */
    struct timeval    vai_down_timer;      /* down timer computed */
                                           /* for master */
} vrrp_adv_info_t;
```

The function vrrp_get_inst() 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.6.2 Retrieve a VRRP group's information

```
vrrp_ret_t vrrp_get_grp(vgrp_name_t name, vrrp_grp_t *grp,
    vrrp_status_t *stat, vrrp_vr_t **vr, vrrp_adv_info_t **adv);
```

The function vrrp_get_grp() works basicly in the same way as vrrp_get_inst() does. The caller should allocate a vrrp_grp_t structure and a vrrp_status_t structure whose pointers are to be passed in as the output parameters grp and stat. The stat parameter could be passed as NULL and then the function will skip returning the status information.

The last two parameters are also output parameters, which are used to get all virtual router information of the group, and the last seen advertisement information for every vritual router. These two parameters could be passed in as NULL, in which case the function will skip returning the information. If they are not NULL, the function will allocate a vrrp_vr_t array and a vrrp_adv_info_t array, and return the array pointers through the output parameter vr and adv. The lengths of the two returned arrays are both indicated by the value of grp.vg_vrnum which is returned through the output parameter grp. If the allocation failed, the function will return VRRP_ENOMEM.

The caller should free the returned vrrp_vr_t array and the vrrp_adv_info_t array if the passed paramters vr and adv are not NULL.

If the required group does not exist, the functions will return VRRP_ENOGRP.

This function doesn't require extra privileges.

A.6.3 Update a VRRP instance's properties

```
vrrp_ret_t vrrp_set_inst_prop(vinst_name_t name,
       vrpri_t priority, int delay, int preempt, int accept);

#define    VR_PROP_UNCHANGE     -1
#define    VR_PROP_PREEMPT      1
#define    VR_PROP_UNPREEMPT    0
#define    VR_PROP_ACCEPT       1
#define    VR_PROP_NOTACCEPT    0
```

The parameters priority, delay, preempt and accept can be VR_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.6.4 Update a VRRP group's properties

```
vrrp_ret_t vrrp_set_grp_prop(vgrp_name_t name,
       vrpri_t priority, int delay, int preempt, int accept);
```

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

This function requires the PRIV_SYS_NET_CONFIG and PRIV_NET_RAWACCESS privileges.

A.7 Look up instances or groups

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

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

These two functions are used to look up existing instances or groups on the system based on VRID, interface name, and addresses 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 or groups matching the given attributes.

The number of found instances or groups are returned in the output parameter num, and the names of all found instances or groups are returned as a "char [][VNSIZ]" array through the ourput parameter names. If the returned num is not 0 and names is not NULL, the caller should free the array.

These two functions don'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/vrrp/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/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>
```

The sub-commands are:

`create`: to create and start running.
`startup`: to send a "startup" event to the instance/group. The instance/group will leave INIT state on receiving it.
`shutdown`: to send a "shutdown" event to the instance/group. The instance/group will enter INIT state on receiving it.
`remove`: to remove from the system.
`Modify:` to modify run-time properties.
`show`: to show properties.

Most of these commands (create, startup, shutdown, remove, 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 or group:

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

```
    <vinst_name>

vrrpadm create -g <vgrp_name>
    [-p <priority>] [-d <adv_interval>] [-o <flags>]
    -v <vrid> -i <intf> [-j <primary_aDdr>] -A <ipaddr_list>
    [-v <vrid> -i <intf> [-j <primary_addr>] -A <ipaddr_list>
```

The options and the corresponding configuration file format will be detailed in Section 4.

3.2 Startup

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

```
vrrpadm startup -g <vgrp_name>
vrrpadm startup -v <vinst_name>
```

3.3 Shutdown

To enter the INIT state from MASTER or BACKUP state:

```
vrrpadm shutdown -g <vgrp_name>
vrrpadm shutdown -v <vinst_name>
```

3.4 Removal

To remove a group or a standalone VRRP instance from the system:

```
vrrpadm remove -g <vgrp_name>
vrrpadm remove -v <vinst_name>
```

3.5 Modification

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

```
vrrpadm modify -g <vgrp_name> \
    [-p <priority>] [-d <adv_interval>] [-o <flags>]
vrrpadm modify -v <vinst_name> \
    [-p <priority>] [-d <adv_interval>] [-o <flags>]
```

All VRRP instances in the group will be affected if "-g" is specified.

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 [-V] [-P]
vrrpadm show -g <vgrp_name> [-V] [-P]
vrrpadm show -v <vinst_name> [-V] [-P]
```

"-V" enables the verbose mode, and "-P" displayes in a machine parsable format. By default it displays in human readable format.

(1) The verbose, human readable display for a standalone instance:

```
INSTANCE vinst4
  vrid = 4
     priority = 100
     adv_intval = 2
     preempt_mode = true
     accept_mode = false
     interface = bge2
     primary_ipaddr = 10.100.77.77
     associated_ipaddrs = 10.100.77.88,10.100.77.99
     previous_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:

```
state = INIT
```

(2) The verbose, human readable display for a group:

```
GROUP vgrp3
    priority = 100
    adv_intval = 3
    preempt_mode = true
    accept_mode = false
    state = BACK
    INSTANCE
      vrid = 7
        [ ... fields specific to this instance ... ]
    INSTANCE
      vrid = 8
        [ ... fields specific to this instance ... ]
```

(3) Machine parsable output is designed for easy parse for scripts. Each VRRP instance, either standalone or in a group, is displayed in a line. For a standalone instance, its group_name is empty (""). For a instance in a group, its inst_name is empty (""). Fields that don't apply are not displayed in this line.

Below is an example. The output is quite long, so in this document we split it into several lines, but they're actually a single output line.

```
group_name="vgrp3" inst_name="" \
   vrid="4" priority="100" \
   adv_intval="2" preempt_mode="true" accept_mode="false" \
   interface="bge2" primarp_ipaddr="10.100.77.77" \
   associate_ipaddrs="10.100.77.88,10.100.77.99" \
   state="BACK" master_priority="100" master_src_addr="10.100.77.78" \
   master_adv_intval="1.00" master_adv_last="1220590518.323" \
   skew_time="0.391" master_down_intval="3.391" \
    master_down_timer="3.068"
```

(4) The above are all for verbose mode, where as much as possible information is displayed. For non-verbose mode, some fields with too much details are not displayed:

```
INSTANCE vinst4
   vrid = 4
    priority = 100
    adv_intval = 2
    preempt_mode = true
    accept_mode = false
    interface = bge2
    primary_ipaddr = 10.100.77.77
    associated_ipaddrs = 10.100.77.88,10.100.77.99
    state = BACK | MAST | INIT

GROUP vgrp3
    priority = 100
    adv_intval = 3
    preempt_mode = true
    accept_mode = false
    state = BACK
    INSTANCE
      vrid = 7
        interface = bge2
        primary_ipaddr = 10.100.77.77
```

```
          associated_ipaddrs = 10.100.77.88,10.100.77.99
      INSTANCE
         vrid = 8
          interface = bge3
          primary_ipaddr = 10.100.78.77
          associated_ipaddrs = 10.100.78.88,10.100.78.99

group_name="vgrp3" inst_name="" \
    vrid="4" priority="100" \
    adv_intval="2" preempt_mode="true" accept_mode="false" \
    interface="bge2" primarp_ipaddr="10.100.77.77" \
    associate_ipaddrs="10.100.77.88,10.100.77.99" \
    state="BACK"
```

4. "vrrpadm create" Command line options

To create a VRRP instance:

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

To create a VRRP group:

```
vrrpadm create -g <vgrp_name>
   [-p <priority>] [-d <adv_interval>] [-o <flags>]
   -v <vrid> -i <intf> [-j <primary_addr>] -A <ipaddr_list>
   [-v <vrid> -i <intf> [-j <primary_addr>] -A <ipaddr_list>
```

Note, in either case, the user must specify a name for the instance or group to be created. The name must be a string consisting of letters, digits and underscores. When creating a group, the VRRP instances in this group don't have names.

-g <vgrp_name>
   To indicate that a group is to be created, and its name.

The following three options are shared by all VRRP instances in the same group:

-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 all will be set to true. Example: -o preempt,not_accept.

The following options are instance-specific:

-v <vrid>:
the VRID. Value range: 1-255.
"-v" should be the first instance-specific options.

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

Example 1: a standalone VRRP instance.

```
vrrpadm create -v 13 -i nge1 -j 11.12.13.90
    -p 200 -A 192.168.10.10,192.168.10.11
    vinst13
```

Example 2: a group with two VRRP instances.

```
vrrpadm create -g vgrp3 -d 1 -o un_preempt
    -v 12 -i nge1 -A 1.2.3.4
    -v 13 -i e1000g1 -j 11.12.13.90 -A 11.12.13.14
```

5. The configuration file

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

The configuration file uses a nested block-structured syntax similar to what is used for gated.conf. The whole configuration file looks like:

```
vrrp_group {
      group_name      vrgp3;
        :
      vrrp_instance {
              vrid    12;
              interface e1000g1;
              :
      }
      vrrp_instance {
              vrid    13;
              interface nge1;
              :
      }
}
```

```
vrrp_instance {
    inst_name  inst2;
    vrid   14;
    interface nge2;
    :
}
```

Let's examine it in more detail.

A VRRP instance is configured as:

```
vrrp_instance {
    inst_name         <vinst_name>;
    active            true | false;
    priority          <priority>;
    advertisement_interval      <adv_interval>;
    preemption_mode          enabled | disabled;
    accept_mode           enabled | disabled;

    vrid              <vrid>;
    interface         <intf>;
    primary_address          <primary_addr>;
    vr_ipaddr_list           <vr_ipaddr_list>;
    ...
}
```

Note, if the VRRP instance is in a group, inst_name and following the five variable are not needed; the group will define these five variables for the whole group; all VRRP instances will share the values, and will silently ignore what are defined in the VRRP instance.

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.

A group is configured as:

```
vrrp_group {
    group_name        <grp_name>;

    # The following five are shared by all VRRP instances in me.
    active            true | false;
    priority          <priority>;
    advertisement_interval      <adv_interval>;
    preemption_mode          enabled | disabled;
    accept_mode           enabled | disabled;

    # Individual VRRP instances in me.
    vrrp_instance {
        ...
    }
    vrrp_instance {
        ...
    }
}
```

Note, the "active" flag has the same semantics as in the standalone instance, but now it applies to all instances in this group.

Example: a configuration file which defines a group with two VRRP instances and a standalone VRRP instance.

```
### Beginning of example configuration file

# A group with two VRRP instances
vrrp_group {
    group_name              vgrp3;
    active                  true;
    priority                200;
    advertisement_interval  2;
    preemption_mode         disabled;
    accept_mode             disabled;

    vrrp_instance {
        vrid                13;
        interface           nge1;
        primary_address     192.168.10.8;
        vr_ipaddr_list      192.168.10.10,192.168.10.11;
    }

    vrrp_instance {
        vrid             12;
        interface        e1000g1;
        vr_ipaddr_list   11.12.13.14;
    }
}

# A standalone VRRP instance
vrrp_instance {
    inst_name               vinst2;
    active                  true;
    priority                225;
    advertisement_interval  1;
    preemption_mode         disabled;
    accept_mode             disabled;

    vrid                    20;
    primary_address         192.168.10.200;
    vr_ipaddr_list          192.168.10.98;
}

### End of example configuration file
```