Brussels II - ipadm and libipadm. *Revision* : 1.12

Sowmini Varadhan, Girish Moodalbail, Vasumathi Sundaram

September 22, 2009

Contents

1 Motivation

As with Phase I, the primary motivation for this project is the recognized well-known inadequacy of `ndd(1m)`, especially in the post-`SMF` world. Primary CRs that capture the problem are:

- 6215036 (http://monaco.sfbay/detail.jsf?cr=6215036) "Stable Interface for Boot-time NDD Configuration"
- 6597312 (http://monaco.sfbay/detail.jsf?cr=6597312) "ip_addrs_per_if not respected during boot; limited to 256 addresses per if"
- 4404812 (http://monaco.sfbay/detail.jsf?cr=4404812) "set network card configuration with ndd-conf.sh"

Essentially TCP/IP parameters are modified only via `ndd(1m)` today. The value range that can be applied on a parameter is not clearly known to the user. They have to skim through the source code to find the value range that can be applied on a parameter. Further these settings are not automatically applied on reboot. Prior to smf, customers were able to (merely) simulate persistence by editing the `/etc/rc.d` scripts, but after the introduction of `smf(5)`, there is less fine-tuning possible over when the parameters may be applied.

In addition, since `ndd(1m)` pre-dates debugging and monitoring tools like `mdb(1)` and `dtrace(1m)`, it has been abused in the days to dump elaborate details about system state. This approach faces its own constraints such as hitting the limits on buffer sizes (See CR 4616660) and being unusable on customer crash-dumps.

Thus one goal of the Brussels project is to provide a new tool, '`ipadm`', that will be an improved alternative to `ndd(1m)`. The core functionality of `ipadm` will be placed in a library '`libipadm`' that will

- export interfaces that can utilized by other programs, including `/sbin/ndd`
- provide persistent settings of TCP/IP tunables.
- provide current/default/possible-values for each TCP/IP tunable.

An additional goal of Brussels Phase II is to lay the foundations for an extensible networking library in `libipadm` that can be used by multiple configuration tools. The existing commands (`ifconfig`, `dladm`, `ipadm` etc.) will be backed up by a library, with the command itself becoming a thin wrapper around the library. The main benefit of this approach is that it provides a solution for a problem frequently encountered by applications trying to plug into the Networking code. A classic instance of this problem is for IP interface plumbing. The actual implementation details are not Public Interfaces, and are available in `ifconfig(1m)` alone. Thus when some application like `nwamd`, or a clustering daemon, needs to plumb interfaces, they are forced to either copy/paste the code, or spawn (i.e., `fork(2)` and `exec(2)`) a copy of `ifconfig(1m)`. The former approach is fragile, since many of these Interfaces are Consolidation/Project Private. The latter approach is inefficient.

In contrast, `libipadm` will provide Stable Interfaces that may be used by third party applications.

The "alternative to `ndd(1m)`" problem will be solved by having `ipadm(1m)` make calls into `libipadm` to set/get TCP/IP tunables in a user-friendly manner.

Brussels II will provide a tool, `/sbin/ipadm`, and an associated library `/lib/libipadm{.so.1}`, that satisfy the following requirements:

- `libipadm` will encapsulate key networking functionality needed at Layer 3 and Layer 4 of IP Networking for Solaris.
- The initial delivery will contain Interfaces to be shared (where possible[1]) between the existing prominent consumers in ON e.g., `ifconfig`, `nwamd`, `pppd` and `rcm_daemon`.
- The initial delivery of ipadm will cover basic features and lay the foundations for future extensions for a more exhaustive feature list. This minimum functionality to be provided in the initial delivery will include

  1. link plumb/unplumb,
  2. address management (add/delete/show),
  3. tuning management (set/get).

  The proposed syntax of these commands is described in Section 2
- `libipadm` should have sufficent generality that it can (and will) be extended for use by other modules.
- A serious drawback in existing tools like `ndd(1m)` today is the absence of support for persistent configuration, whereby a setting is repeatedly applied on each restart of the targetted sub-system. The `libipadm` solution will provide a well-defined option for persistent configuration where in persistent tunable settings made via ipadm will be automatically reapplied on reboot, before any other networking application is started.. The details of this solution are described in Section 2.
- provide MDB dcmds as appropriate for debugging kernel structures via mdb (formerly done using `ndd`) and remove the corresponding `ndd` support code in the kernel.
- `ndd` compatibility: `/sbin/ndd` will be converted to use the interfaces provided by `libipadm`, and the implementation of these interfaces may themselves have Consolidation Private stability. However, since there may be applications that directly execute `ND_SET` and `ND_GET` ioctl calls minimum support for these ioctls will be provided for transitionary support of these applications.
- the syntax for invoking ipadm will follow the $< verb > - < object >$ convention introduced by PSARC 2006/406 [7], and followed in `/sbin/dladm` and `/sbin/flowadm`.
- there will be a 1-1 mapping between an ipadm invocation and a library interface. For example, `/sbin/ipadm verb-obj <..>` will correspond to a `ipadm_verb_obj()` library interface). In addition, all commands will follow the existing conventions used in `/sbin/dladm` and `/sbin/flowadm` for parsable output, column selection. Where it is possible, the functionality for processing command-line input will be extracted to a common shared library (See CR 6782154).
- libipadm will supercede the Consolidation Private interfaces provided by libinetcfg, and libinetcfg invocations in ON will be converted to use libipadm.

2 ipadm(1m) CLI

This section proposes a command-line interface for `/sbin/ipadm`. All ipadm subcommands, except show-*, will support a `-t` flag for temporary version of the command, so that the applied change will not persist across reboot.

For all the `show-*` subcommands described in this document, the `-p or -c` and `-o` flags provide parsable output support that is compatible with comparable support in `dladm, flowadm, ipmpstat etc.` commands, and may be used to process the information printed above through other standard shell filters.

[1]Note that the primary purpose of introducing `libipadm` is to provide more stream-lined, efficient Interfaces that can be supported viably in a Stable manner. Thus, if the conversion of an existing consumer requires compromises to the design of the Interface, priority will be given to keeping the design of `libipadm` clean, and the conversion of existing consuers will be done on a "Best Effort" basis.

3 Interface management

3.1 Creating the physical interface

```
# ipadm create-if [-t] [-f inet | inet6] <interface>
```

The "`create-if`" subcommand plumbs an IP interface and does not need to be explicitly invoked in the typical case(e.g., prior to `create-addr`). The plumbing functionality is encapsulated in `ipadm create-if` (and its library counterpart `libipadm‘ipadm_create_if()`) for applications that wish to plumb the interface but do not expect to explicity configure an address on the interface.

If the `-f` option is specified then the created interface will only handle packets of the IP version type specified by the '`-f`' option. The address of the plumbed interface will be set to INADDR_ANY (for '`inet`') or to `::` for IPv6.

The default behavior for `create-if` (without the `-t` option) is to consult the persistent database and apply any configuration that applies to `interface`. In conjunction with the `-t` flag, the `create-if` subcommand operates in the "ifconfig" mode, i.e., the interface is plumbed, but the persistent database is never consulted to retrieve any configuration for the object that may have been stored earlier.

3.2 Destroying the physical interface

```
# ipadm delete-if [-t] [-f inet | inet6] <interface>
```

Unplumbs the interface handling the IP version type specified by the '`-f`' option (both IPv4 and IPv6, if no `-f` is specified). All addresses configured on that interface will be torn down.

By default, this subcommand flushes any references to the interface from the peristent store so that the interface will not be recreated on reboot. If, however, the `-t` option is used, the `delete-if` subcommand operates in the "ifconfig" mode, where the interface is removed from the running configuration but the persistent storage is left intact.

3.3 Modifying properties of the interface

The subcommands

```
# ipadm set-ifprop [-t] [-f inet | inet6] -p <prop>=<value>[,..] <interface>
# ipadm show-ifprop [-P] [[-c]-o <field>[,...]] [-p <prop>,...] [<interface>]
# ipadm reset-ifprop [-t]  [-f inet | inet6] -p <prop> <interface>
```

may be used to set/get/reset per interface properties. For the `set-ifprop` and `reset-ifprop` subcommand only one property can be specified at a time. The `reset-ifprop` subcommand will reset the given property to the values it had at the startup. Also see Section 15.1 for a discussion on the relation with global properties.

The list of supported interface properties and their description can be found in `ipadm` man page at [3].

3.4 Display Interface Status

The interface status in the active configuration can be viewed through the ipadm subcommand

```
# ipadm show-if [-P] [[-p] -o <field>[,...]] [<interface>]
```

When `-P` is specified, it shows the persistent configuration for the given interface stored in the database.

3.5 Examples

Sample output is shown below[2]:

INTF	MTU	STATE	FLAGS
lo0	8232	ok	mcast,virtual
e1000g0	1500	ok	bcast,mcast
e1000g1	1500	down	bcast,mcast,router
ipmp0	1500	failed	bcast,mcast,ipmp
qfe0	1500	down	bcast,mcast
tun0	1480	down	mcast,pointopoint,noarp
vni0	68	ok	noarp,noxmit,virtual
bge0	9000	ok	mcast,nortexchg,nonud

Managing interface properties:

```
# ipadm set-ifprop -p mtu=900 net0      # mtu (for both v4/v6) is set to 900
# ipadm set-ifprop -f inet6 -p mtu=1300 net0 # mtu for v6 interface is set to 1300
# ipadm set-ifprop -p forwarding=yes net0   # enables forwarding on net0
# ipadm set-ifprop -p reasm_timeout=60 net0 # reassembly fragment timeout on
      net0 is set to 60 seconds

# ipadm show-ifprop -p mtu,forwarding,metric,reasm_timeout net0
```

INTF	PROPERTY	PROTO	FAMILY	PERMS	VALUE	DEFAULT	POSSIBLE
net0	mtu	ip	inet	rw	900	1500	68-9000
net0	mtu	ip	inet6	rw	1300	1500	68-9000
net0	metric	ip	inet	rw	2	0	–
net0	metric	ip	inet6	rw	2	0	–
net0	reasm_timeout	ip	inet	rw	60	15	1 - 6000
net0	reasm_timeout	ip	inet6	rw	60	60	1 - 6000

```
# ipadm reset-ifprop -p mtu net0 # resets the mtu to default value for net0
```

4 Address Creation

The `create-addr` and `delete-addr` subcommands are used to manage static IPv4 and IPv6, DHCP and IPv6 addrconf addresses on interfaces, and are described in this section.

The general synopsis of the `ipadm create-addr` subcommand is

```
# ipadm create-addr -T {static | dhcp | addrconf}  <type-specific args> <addrobj>
```

Each of the types, `static, dhcp, addrconf`, is described in further detail below.

All addresses configured on an interface are collectively identified by an address object, `addrobj`. An `addrobj` is of the form "`interface/user_specified_string`" where

[2]Note that lo0 has different MTU values for IPv4 and IPv6 interfaces due to an implementation artifact, and the displayed value would be the lower of IPv4 and IPv6 loopback mtu

- the `interface` is the name of the IP interface on which the address is configured, and
- `user_specified_string` is a string of alphanumeric characters and can be at-most 32 characters in length and must begin with an alphabet.

Uniqueness of the interface name ensures that `addrobj` values identify a unique address or set of addresses on the system. The `addrobj` must be specified as the object of all `*-addr` subcommands. The `interface` component of the `addrobj` will be referred to as the "implied interface" for the `addrobj` in the remainder of this document.

4.1 Add static address

```
# ipadm create-addr [-t] -T static [-d] -a {local|remote}=<addr>/[<prefixlen>],... <addrobj>
```

Configures address in `local=<addr>/<prefixlen>` as the local IPv4 or IPv6 address on the implied interface for `addrobj`. The `<addr>` may be any of the following:

1. a numeric address in CIDR notation, specified as `addr/prefixlen`. If the `prefixlen` is not specified, the netmask is computed using the order specified for the `netmasks` line `/etc/nsswitch.conf` or using Classful Address semantics.
2. a hostname whose numeric value is uniquely obtained using the resolver order specified for `hosts` or `ipnodes` in `nsswitch.conf`. As with numeric addresses, if the netmask is not specified, it is computed using `nsswitch.conf` or classful address semantics

When hostnames are specified, the name resolution **must** return exactly one numeric address in order for the `create-addr` to succeed. Otherwise an error will be generated.

The address in `remote=<addr>` specifies the address of the remote end-point of point-to-point interfaces. It may be specified as a numeric address or hostname, following the same rules as for the `laddr`, but any netmask provided will be ignored.

The implied interface for `addrobj` will be plumbed if necessary by the `create-addr` invocation. Note that attempting to create a non-link-local IPv6 address on an interface that does not have a link-local address will result in an error. The link-local address will have to be generated either statically (by invoking `create-addr` and specifying all 128 bits of the address) or by using Stateless/Stateful Address configuration, using (see Section 4.3).

When `ipadm create-addr` is used to set up a static IPv6 address on an interface, the `in.ndpd` daemon is notified to **not** auto-configure IPv6 addresses on the interface unless it has been explicitly requested to do so by the methods described in Section 4.3. Note that the default link-local may be generated without turning on IPv6 ADDRCONF by following the steps show in Section 4.3.1

4.2 Examples

```
# ipadm create-addr -a local=12.1.2.3,remote=12.1.2.4 tun0/mytunaddr
# ipadm create-addr -a local=11.22.33.44/24 net0/netaddr
```

The address is enabled for use as a source/destination of outbound/inbound packets by default. However, this default may be modified by specifying the `-d` option, in which case the address is configured on the system but will not be enabled for use as a source/destination of IP packets.

This subcommand is persistent by default, i.e., an address added by '`ipadm create-addr <...>`' updates the persistent store, and will be recreated on reboot. If the addresses were specified as hostname during `create-addr`, restoration of the address will re-consult the name-resolver routines and apply the most recent numeric address that is returned by the resolver. Note the resolver is consulted only once (during address creation), and subsequent address management routines cache and use the numeric address obtained for other `addr` commands.

The '`-t`' flag can be used for temporary changes.

4.3 IPv6 Stateless/Stateful address management

Groups of addresses defined by IPv6 Stateless/Stateful address configuration are created by using the `-T addrconf` option with `create-addr` and deleted using the `delete-addr` command.

The command

```
# ipadm create-addr [-t] [-I interface_id] \
                [-p {stateless|stateful} = {yes|no},...] <addrobj>
```

creates autoconfigured IPv6 addresses on the interface implied by `addrobj`.

The `interface_id` is the Interface ID (aka token) to be used for generating auto-configured addresses. If it is not specified, the default Interface ID is generated from the MAC address of the ethernet interface. Currently, only one Interface ID is supported per interface. Once a group of addresses is created using the default or the user-supplied Interface ID, the ability to create a new group of addresses with a different Interface ID is not supported.

By default, IPv6 addresses will be autoconfigured based on prefixes advertised by routers as described in RFC 4862. When `stateless=no` is specified, stateless autoconfiguration based on advertised prefixes will not be performed.

By default, DHCPv6 will be used to configure IPv6 addresses, as described in RFC 3315. DHCPv6 will be automatically invoked by in.ndpd based on the "managed" flags sent by routers. DHCPv6 is disabled when `-p stateful=no` option is specified.

When the interface is created using ipadm and a static link-local address assigned, no stateless or stateful autoconfiguration will be performed on the interface by in.ndpd until `create-addr` is invoked with the `-T addrconf` type.

The default IAID for DHCPv6 is the ifIndex number of `ifname` and is kept in stable storage to ensure consistency of the (ifname, IAID) pair. If this default is unavailable, arbitrary number will be generated. Future extensions to the `create-addr -T addrconf` command will allow a custom IAID to be provided via the `-I` option.

The default DUID for DHCPv6 is LLT if possible, otherwise we use the UUID library to generate an arbitrary one. Future extensions to the `create-addr` command will allow a custom DUID to be supplied using the `-C` option.

When `ipadm create-addr -T addrconf` is used to create addresses, in.ndpd is sent a msg to start the autoconfiguration and is provided with the interface name and the interface ID to use. The daemon in.ndpd then starts sending the Router Solicitations. On receipt of the Router Advertisements, it creates addressses for the received prefixes using the interface ID that was sent by ipadm, depending on whether stateless and stateful autconfiguration were enabled through the options `stateless` and `stateful` respectively.

If the interface variables `StatelessAddrConf` and `StatefulAddrConf` are initialized in ndpd.conf(4) file, and if `ipadm create-addr -T addrconf` is used, the values provided for options `stateless` and `stateful` will override those of variables `StatelessAddrConf` and `StatefulAddrConf`. The `show-addr` command can be used to view the addresses associated with a given `addrobj`.

4.3.1 Examples

The example below shows some addresses obtained by stateless address configuration:

```
# ipadm create-addr -T addrconf -I ::abcd/64 link0/v6addrs
```

The field `PFXORIGIN` describes the origin of the Prefix (see RFC 4293, Page 15). For a description of `FLAGS` and `STATE`, refer to Section 5.

The example below would generate the default link-local on link0 without enabling stateless or stateful address configuration

```
# ipadm create-addr -T addrconf -p stateless=no,stateful=no link0/llonly
```

Address state as printed by ipadm	Address state flag returned by ipadm_show_address	Semantics
deprecated	IFA_DEPRECATED	similar to IFF_DEPRECATED
invalid	IFA_INVALID	Address is down. Corresponds to ~IFF_UP
inaccessible	IFA_INACCESSIBLE	Interface is down, e.g., ~IFF_RUNNING
unknown	IFA_UNKNOWN	Status unknown, because e.g., interface is missing (Section 17)
tentative	IFA_TENTATIVE	DAD has been initiated
duplicate	IFA_DUPLICATE	DAD failure
preferred	IFA_PREFERRED	Default value in the absence of other qualifiers

Table 1: STATE information returned by ipadm show-addr

4.4 IPv4 DHCP address management

The command

```
# ipadm create-addr [-t] -T dhcp [-w wait_time] [-p] <addrobj>
```

creates a dhcp-controlled IPv4 address on the implied interface for the `addrobj`. The default client-id for DHCP is constructed by the system. Future extensions will allow a custom `client-id` to be supplied using the -C option.

The `-w` option is used to specify the maximum amount of time to wait before the command completes. A wait_time value of `forever` is used to block until the command completes. The option `-p` specifies if this is the primary dhcp interface.

4.4.1 Example

The example below shows an address obtained by DHCP.

```
# ipadm create-addr -T dhcp -w forever -p net0/myv4addr
```

5 Displaying IP address on the system

The `/sbin/ipadm` invocation:

```
# ipadm show-addr [-P] [<addrobj>]
```

will display all the IP addresses available on the system or the address identified by `<addrobj>`. If `-P` is specified, this will display the addresses from the persistent configuration stored in the database. The information will contain the following fields:

- ORIGIN, which desribes the origin of the address (see RFC 4293, Page 14) and will be one of `{addrconf, static, dhcp}`. Here, `addrconf` indicates that the address was obtained by Stateless/Stateful Address Configuration. The `dhcp` indicates that the IPv4 address was configured using DHCP.
- STATE, which descibes the Address status and may be one of the states desribed in Table 1

5.1 Examples

```
  # dladm show-iptun
  LINK            TYPE  FLAGS  SOURCE              DESTINATION
  tun0            ipv4  --     10.8.48.242         10.8.57.1

 # ipadm create-addr -T static -a local=10.8.48.242,remote=10.8.48.1 tun0/tunaddr2
 # ipadm create-addr -a local=11.2.3.1/8 vni0/vniaddr
 # ipadm show-addr
```

OBJECT	ORIGIN	STATE	FLAGS	ADDR
lo0/loopback4	static	preferred	---	127.0.0.1/8
vni0/vniaddr	static	preferred	---	11.2.3.1/8
bge0/dhcpaddr	dhcp	preferred	---	10.8.48.173/25
bge1/dhaddr	dhcp	preferred	---	10.8.48.242/25
tun0/tunaddr2	static	preferred	U--	10.8.48.242-->10.8.48.1
lo0/loopback6	static	preferred	---	::1/128
bge0/dhcp6addr	addrconf	preferred	---	fe80::203:baff:fe44:3d34/10
bge0/dhcp6addr	addrconf	preferred	---	2002:a08:39f0:3:203:baff:fe44:3d34/64
bge1/auto	addrconf	preferred	---	fe80::203:baff:fe44:3d35/10
bge1/auto	addrconf	preferred	---	2002:a08:39f0:3:203:baff:fe44:3d35/64

```
Possible values of flags:
  U: Unnumbered    p: Private  t: Temporary
```

6 Deleting an address on an interface

The command

```
# ipadm delete-addr [-t] <addrobj>
```

removes the address identified by `<addrobj>` on the implied interface. Argument semantics are the same as those described in Section 4.1. Note that name-resolution routines are **not** consulted for deleting the address.

This command is persistent by default, i.e., an address added by '`ipadm delete-addr <...>`' updates the persistent store. The '`-t`' flag can be used for temporary changes. Attempting to persistently delete an address when either the address or the interface has been created with the '`-t`' option will generate a warning about the missing update to the persistent store, but the address will be deleted (if it exists). Similarly, attempting to temporarily delete an address (i.e, with the `-t`) on an interface or address that has been created persistently will generate a warning about the inconsistent operation on the persistent store.

Due to implementation details of the Solaris kernel, there are some complications with deleting the first address created on an interface. The issues raised by this constraint and proposed solution are discussed in Section 6.1

6.1 Subtleties in address deletion

As mentioned earlier, the implementation choices made in the Solaris kernel impose some contraints on address deletion using the existing IP ioctls. Specifically, while the first address configured on an interface may not be deleted without unplumbing the interface and tearing down all other addresses, though all the other (second and later) addresses may be removed using SIOCSLIFREMOVEIF.

The existing infrastructure only permits the first address to be marked ~IFF_UP, i.e., administratively disabled. This in turn, leads to other complexities in the view presented to user-space. Applications compiled for other OS'es like BSD interpret the absence of IFF_UP on a link `net0` as "the link is down", and would arrive at the conclusion that all addresses on that link are unavailable for these OS'es. On Solaris, these applications have to actually walk the list of available addresses before making the same conclusion.

However, even the acquisition of the list of addresses on an interface is not easily available on Solaris. While other OS'es like BSD and Linux provide support for the `getifaddrs()` library call [1], Solaris only provides the harder-to-use SIOCGLIFCONF ioctl.

A detailed discussion of the problem and some long-term solutions are available in [5]. As described in that proposal, `ipadm` and `libipadm` will emulate deletion of the first address by replacing it with INADDR_ANY or `::`. In addition

- the `getifaddrs()` command will be ported to Solaris and made available through `libipadm`. The input/output semantics will be the same as that described in [1], except that the interface name will be a logical interface name (e.g., `net0:5`) that can be used in other IP ioctl calls. The reported address information may contain `AF_LINK` addresses in the future, so that the caller will not make assumptions about the address type. Only ~IFF_UP addresses will be reported.
- `ipadm_addr_info()` will return IP address information for all IP addresses on the system. This will include any address that is ~IFF_UP, excluding the INADDR_ANY or `::` holes left from the deletion of the first address. Additionally, `ipadm_addr_info()` will also report ORIGIN and STATE as described in Section 5. The interface name returned by the library call will be the logical interface name.

7 Refreshing IP Addresses

A common command

```
ipadm refresh-addr <addrobj>
```

is provided to re-verify the IP address(es). If `addrobj` refers to a static address, then DAD will be restarted (if necessary) on the static address. If `addrobj` refers to a dhcp-controlled address, then the lease duration obtained on the address will be extended by the DHCP client daemon. If `addrobj` refers to a IPv6 autoconfigured address, `in.ndpd` will re-validate the lifetime for the associated prefixes.

8 Enabling IP Address

The ipadm subcommand

```
# ipadm enable-addr <addrobj>
```

is provided to enable IP address identified by `addrobj`. If the `addrobj` is of the type 'static' then the address identified by the address object is enabled, so that it can be used as a source/destination of outbound/inbound packets. This command has no effect if the `addrobj` has been disabled by the system because it is a duplicate address, or if the address was enabled prior to the `enable-addr` invocation. Since they are managed by other protocols, DHCP and IPv6 auto-configured addresses may not be enabled by this command, and an error is generated if `<addrobj>` is of type 'dhcp' or 'addrconf'.

9 Disabling IP Address

The ipadm subcommand

```
# ipadm disable-addr <addrobj>
```

is provided to disable IP address identified by `\<addrobj\>`. If the `\<addrobj\>` is of the type `static` then the address identified by the address object is disabled, so that it cannot be used as a source/destination of outbound/inbound packets. This command has no effect if the `\<addrobj\>` was already disabled prior to the `disable-addr` invocation. Since they are managed by other protocols, DHCP and IPv6 auto-configured addresses may not be disable by this command, and an error is generated if `<addrobj>` is of type 'dhcp' or 'addrconf'.

10 Address Properties

The `ipadm set-addrprop` and `show-addrprop` subcommands may be used to tune interface address properties. The syntax of the commands are:

```
# ipadm set-addrprop [-t]  -p <prop>=<value>[,...] <addrobj>
# ipadm show-addrprop [-P] [[-c] -o <field>[,...]] [-p <prop>,...] <addrobj>
```

The `ipadm set-addrprop` sets the value of a property on the `<addrobj>` specified. If the `<addrobj>` maps to several addresses, then property changes applies to all the addresses referenced by the ¡addrobj¿. Only one property can be specified at a time.

The `ipadm show-addrprop` shows the current, default and possible range of one or more properties.

The supported address properties are `prefixlen, private, transmit and zone`. For more information on these properties, please refer to the `ipadm(1m)` man page [3].

10.1 Examples

```
# ipadm show-addrprop bge1/v4addr
OBJECT              PROPERTY     VALUE    DEFAULT  POSSIBLE
bge1/v4addr         prefixlen    8        8        1-32
bge1/v4addr         private      off      off      on,off
bge1/v4addr         transmit         on       on       on,off
bge1/v4addr         zone         global   global   --

# ipadm set-addrprop -p prefixlen=24 bge1/v4addr
# ipadm set-addrprop -p private=on bge1/v4addr
# ipadm show-addrprop -p perfixlen,private bge1/v4addr
OBJECT              PROPERTY     VALUE    DEFAULT  POSSIBLE
bge1/v4addr         prefixlen    24       8        1-32
bge1/v4addr         private      on       off      on,off
```

11 Global Property/tunable management

The subcommands

```
# ipadm set-prop [-t] -p <prop>=<value>[,...] <protocol_module>
# ipadm reset-prop [-t] -p <prop> <protocol_module>
# ipadm show-prop [-P] [[-c] -o <field>[,...]] [-p <prop>[,...]] [<protocol_module>]
```

provide TCP/IP tunable management equivalent to the features of the `/sbin/ndd` command to be applied on the appropriate `protocol_module` (currently tcp, ip, udp and sctp.).Details of each of these sub-commands are provided in the sections below. The subcommands themselves support the setting/viewing of property values that are global to the protocol module itself. Note that in some cases it may be possible to set the value of the property both globally, and on a per-interface basis. with the per-interface value over-riding the global setting. In such cases, the per-interface value must be set using the `set-ifprop` subcommand described in Section 15.2

11.1 Legacy support of ndd tunables

The `ndd(1m)` utility has been changed to use the Consolidation Private interfaces exposed by `libipadm` library. The `ndd` continues to work as it used to before and supports all the properties it used to support, though a warning will be printed if an attempt is made to use `ndd(1m)` to modify the tunables following protocols supported by ipadm (ip, tcp, udp and sctp), notifying the user about the availability of `ipadm` as the preferred tool for property configuration.

In order to improve the usability, some property names have been sanitized to user-friendly versions (e.g., "forwarding" instead of `ip_forwarding`). However it should be noted that `ndd(1m)` will recognize the older version of the property name.

11.2 Setting properties

The subcommand

```
# ipadm set-prop [-t] -p <prop>=<value>[,...] <protocol_module>
```

sets the value of a property on the protocol_module specified. Only one property may be specified in each invocation.

11.3 Resetting properties

The subcommand

```
# ipadm reset-prop [-t] -p <prop> <protocol_module>
```

resets the property to the values it had at the startup. Only one property may be specified in each invocation.

11.4 Getting properties

The subcommand

```
#ipadm show-prop [-P] [[-c] -o <field>[,...]] [-p <prop>[,...]]
   [<protocol_module>]
```

shows the current, default and possible valuse for a given set of properties.

11.5 Use cases and examples

```
# ipadm set-prop -f inet -p ttl=64 ip    # sets the 'ttl' to 64 on 'ip' module.
# ipadm set-prop -f inet6 -p hoplimit=100 ip # sets the 'hoplimit' to 100
# ipadm set-prop -p reasm_timeout=60 ip # sets the reassembly fragment timeout for
                                        all ip interfaces (including those to be
                                        created in the future)

# ipadm show-prop
 PROTO  PROPERTY       FAMILY PERMS      VALUE   DEFAULT   POSSIBLE
 ip     forwarding     inet   rw         on      off       on,off
 ip     forwarding     inet6  rw         on      off       on,off
 ip     ttl            inet   rw         64      255       1-255
 ip   hoplimit  inet6  rw    100     255       1-255
 ip     reasm-timeout  inet   rw         60      15        1-6000
 ip     reasm-timeout  inet6  rw         60      15        1-6000
  :
```

Some use cases are described below:

- attempting to set a property without specifying -f option would mean the value applies for both V4 and V6 packets.
- attempting to set `forwarding` on `tcp` module will generate an error,
- attempting to `set-ifprop -p <module_property>` will generate an error,

12 IPMP sub-commands

Although it will not be delivered as part of the deliverables for the first component of PSARC 2009/306, `ipadm(1m)` will support a family of `*-ipmp` sub-commands that will be delivered as a separate component of the umbrella case for PSARC 2009/306. Soecific details of the structure of the sub-commands will be provided when the component is delivered, but the initial proposal for these commands is available in [2].

13 System call details for property management

`ndd(1m)` uses Mentat's ND_SET and ND_GET system calls to extract property information from the kernel. These system calls are undocumented, Project Private, and extremely fragile (see, for example CR 6567083). The implementation of this code makes several assumptions about the placement of data and white-spaces that could easily be broken by new code.

It would be preferable to have cleaner system-call interfaces to set and get properties. Two socket ioctl's are defined to set/get property values. They being properties, and one to get properties. The `SIOCSETPROP` ioctl to set property value and tThe `SIOCGMODPROP` ioctl to get property value.

Following structure will be used to send/retrieve information from the kernel.

```
typedef struct mod_ioc_prop_s {
        uint_t          mpr_version;
        uint_t          mpr_flags;
        char            mpr_ifname[LIFNAMSIZ];   /* interface name */
        uint_t          mpr_proto;   /* protocol module */
        sa_family_t     mpr_af;   /* IPv4 or IPv6 property */
        char            mpr_name[MAXPROPNAMELEN]; /* property name */
        uint_t          mpr_valsize;   /* size of the buffer */
        char            mpr_val[1];   /* property value itself */
} mod_ioc_prop_t;
```

Unfortunately, even though they are not documented, the existing ND_SET and ND_GET ioctls may be directly invoked from C programs, typically routing daemons like Quagga. Research has indicated that the only such use cases are done for the purpose of setting/getting ip_forwarding and ip6_forwarding global variables, and legacy support for ND_* ioctls to get/set this subset of tunables will be retained in Solaris.

14 ipmgmtd daemon and IP interface management SMF Service

14.1 ipmgmtd

This project adds a new daemon `ipmgmtd` which keeps track of internal state of the library `libipadm` and also provides a means to update the `ipadm` persistent/temporary data-store.

The communication between the library `libipadm` and the `ipmgmtd` is through `door_call(3c)`.

The ipmgmtd deamon performs the tasks listed below.

- The daemon tracks the mapping between `addrobj` and the associated address information, such as interface name, address type, address data (Interface ID, address value etc.).
- The daemon writes these mappings to a file in /etc/svc/volatile/ipadm/ as well, so that it can recover from crashes or restarts.
- Where applicable, the deamon updates persistent data-store with the IP interface configuration and TCP/IP property values changes, to be applied when the interface is restarted.

The daemon runs as a user `netadm` (introduced by NWAM) with basic privileges and does not need any other privileges outside the basic set.

The daemon is controlled by a SMF service as detailed in the next section.

14.2 IP interface management SMF Service

A new SMF service `svc:/network/ip-interface-management` has been created to start the `ipmgmtd` daemon. The daemon has to be started before any other networking service which configures IP interfaces to ensure that the persistent configuration of IP interfaces may be reapplied correctly, while also tracking any state associated with the active configuration. The mechanism for restoring persistent properties is described in Section 15.1.

The `svc:/network/loopback` is the first networking service to configure IP interfaces (loopback, in this case) and also happens to be the first networking service that should be brought up. Thus `svc:/network/loopback` SMF service is dependent on `svc:/network/ip-interface-management`.

The `ip-interface-management` service is not dependent on any other service.

14.3 Exclusive-IP zone support

In each exclusive-IP zone, the `svc:/network/ip-interface-management` SMF service will be enabled and will start the `ipmgmtd` daemon. The daemon will manage the IP interfaces and TCP/IP tunables for that specific zone.

15 Mechanism for restoring Persistent information

There are two aspects to restoring stored configuration information:

1. re-application of `ndd`-like configuration (that applies globally to TCP/UDP/IP/ICMP modules, described in Section 15.1,
2. re-application of interface configuration, including IP addresses, per-interface properties etc., described in Section 15.2

15.1 Persistence of properties global to all interfaces

Although we have some simplicity compared to the dladm-driver interaction because IP is never unloaded, there is still the complexity of **when** to set parameters. In the `rc*.d` model, this was trivially determined by where the `ndd/ifconfig/` etc. invocation was placed in the `rc*d` sequence, but the `smf` model is driven by dependancies and is thus less deterministic. There are some parameters such as `tcp_smallest_anon_port` whose effect can vary widely depending on when the saved setting is restored. There have been many discussions about various solutions for this problem [4].

Ideally we would be able to read the relevant smf settings from `ip_ddi_init()` itself, but as has already been encountered in Brussels in the past, there are no Stable Interfaces to pull `smf` settings into the kernel. The `dlmgmtd` approach used in Brussels I is not a viable solution as `ip_ddi_init` is called before any daemons are spawned.

However, since Networking applications first need to open sockets to be affected by TCP/IP tunable settings, and the ability to open sockets is controlled by `soconfig(1m)`, we propose to create a new process `netstart` that will be started from `inittab(4)` before svc.startd and will do the following:

1. Start `soconfig`
2. Execute `ipadm init-prop` to restore globally persistent settings that apply to all interfaces

Note that the `init-prop` command is only present to initialize global properties during boot, and will not take an `interface` option. The per-interface property settings will be restored when the interface is created as part of `ipadm create-if`.

15.2 Restoration of interface configuration

A rudimentary mechanism for providing persistent configuration of interfaces exists through the files in `/etc` such as `/etc/hostname.intf`. The introduction of `ipadm` is intended to provide an improvement over these existing crude mechanisms. The transition strategy being proposed is:

- When older methods such as `/etc/hostname.<..>` files are detected, these will be applied first.
- Query and apply persistent `ipadm` configuration: the `net-physical` script will first query the `ipadm` persistent store for known interfaces using the `ipadm show-if -P` command, and recreate these addresses (and associated address properties).

If `network/physical` encounters both `/etc/hostname.intf` as well as ipadm configuration information for an interface, the ipadm configuration information will be ignored (after emitting a warning to `/dev/msglog`).

A subsequent (to be added and enhanced, as `ipadm` acquires compatibility with `ifconfig` options) smf service will update existing `/etc/hostname.*` configuration information to input in the `ipadm` store for the next reboot, thereby obsoleting `/etc/hostname.*` file usage.

Analogous principles will be implemented in `network_rcm`: the `/etc/hostname` files will be processed first, followed by re-application of persiste ipadm configuration information.

16 Security Considerations

16.1 prof_attr(4)

We will be using the existing 'Network Management' profile which is already defined in prof_attr(4) database. The purpose of this profile, as defined in /etc/security/prof_attr/ is to "Manage the host and network configuration", which clearly fits our needs of network interface configuration.

16.2 auth_attr(4)

None of the authorizations defined in /etc/security/auth_attr seem to suffice for us. They are mostly related to nwam, reading/writing /etc/hosts, link security and wifi configuration. We would like to introduce the following authorization.

- "solaris.network.interface.config"

  Allows configuring network interfaces (verified in library, libipadm.so.1)

Note: For viewing the interface configurations we don't need any authorization, as ordinary users can already read the interface configuration.

16.3 exec_attr(4)

`ipadm` would need 'sys_ip_config' privilege to configure system's IP interfaces and to configure network parameters/tunables. So we would update exec_attr, for the profile Network Management, as shown below.

```
Network Management:solaris:cmd:::/sbin/ipadm:euid=netadm;egid=sys;privs=sys_ip_config
```

17 Ipadm futures

As pointed out on the OpenSolaris mailing list [6] a current constraint (in both `dladm` and `ipadm`) is the requirement that the interface object being acted upon by any command must exist for the first invocation of the command to exist. Thus one may not be able to add a link to an aggregation if the link is not currently available on the system, or set the mtu of a link that is currently missing.

A long term direction for all the administrative tools is to allow the creation of virtual configuration for objects that may currently be missing but could be added in the future. If this facility were available it would be possible to apply a persistent change on a temporary object. For example, the sequence of commands

```
ipadm create-if -t net0
ipadm create-addr -a local=10.1.2.3/24 net0/staticaddr
```

would appear as the configuration of `10.1.2.3/24` on a missing `net0` link.

The current `ipadm set-prop` interface only permits the setting of tunables that apply globally to all TCP/UDP/IP etc. connections. This should be enhanced to permit fine-tunning of the selected connection by allowing the library/CLI to specify source/destination addresses and ports. Providing this feature will allow binary applications to use new features (e.g., tweak TCP congestion control parameters) without requiring recompilation with new system calls like `setsockopt`.

Another future direction is the ability to support multiple tokens (or Interface IDs) per interface. Currently, each IP interface allows only one token to be set, i.e., once the command `create-addr -T ipv6` is called to create a group of IPv6 addresses for a given Interface ID, it cannot be called again with a different Interface ID. An expected behavior in the future will be to create multiple groups of IPv6 addresses per interface, identified by the addrnames, each corresponding to the Interface ID specified in the `create-addr` command.

18 Acknowledgments

Many thanks to all the team members, and other reviewers who have provided input including:

James Carlson, Honsing Cheng, Renee Danson, Prasanna Kunisetty, Alan Maguire, Peter Memishian, Dave Miner, Dan McDonald, Erik Nordmark, Anders Persson, Darren Reed, Sebastien Roy, Rao Shoaib, Cathy Zhou.

References

[1] "FreeBSD Man Pages: GETIFADDRS(3)"
. http://www.freebsd.org/cgi/man.cgi?query=getifaddrs.

[2] http://arc.opensolaris.org/caselog/PSARC/2009/306/inception.materials/brussels2_design.pdf. "Brussels II - ipadm and libipadm".

[3] "ipadm(1M) man page". http://opensolaris.org/os/project/brussels/Documentation/ipadm_man.

[4] "Setting ndd parameters"
. http://www.opensolaris.org/jive/thread.jspa?messageID=24434&.

[5] "`GLIFCONF/IFF_UP`"
. http://mail.opensolaris.org/pipermail/brussels-dev/2009-May/001527.html.

[6] Erik Nordmark. "Brussels II property for missing interfaces"
. http://mail.opensolaris.org/pipermail/brussels-dev/2009-April/001453.html.

[7] PSARC/2006/406. "Extending dladm for WiFi: An administrative Overview" .
/net/sac.eng/export/sac/PSARC/2006/406/commitment.materials/dladm-wifi.pdf.