Brussels II - ipadm and libipadm. *Revision* : 1.7

Sowmini Varadhan, Girish Moodalbail, Vasumathi Sundaram

May 27, 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 tunable only via `ndd(1m)` today, but 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.

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`, `flowadm`, `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 status 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`, `flowadm`, `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. The details of this solution are described in Section 2.
- Persistent tunable settings made via ipadm should be automatically reapplied on reboot, before any other networking application is started.
- provide macros as appropriate for debugging kernel structures via mdb/dtrace (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 parseable 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.

[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.

2 ipadm(1m) CLI

This section proposes a command-line interface for `/sbin/ipadm`. All commands will support a `-t` flag for temporary version of the command, so that the applied change will not persist across reboot. Note that applying a persistent change that relies on the result of a temporary command may fail with an error.

For all the `show-*` commands described in this document, the `-P` 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.

3 Interface management

3.1 Creating the physical interface

```
# ipadm create-if [-f {inet, inet6}]  <interface>
```

The "`create-if`" plumbs an IP interface and does not need to be explicitly invoked in most cases (e.g., prior to `create-addr`, `create-dhcp`). 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` flags is specified, the created interface will handle packets of the IP version type specified by the '`-f`' flag. Otherwise two interfaces, one for IPv4 and the other for IPv6 will be created. The address of the plumbed interface will be set to INADDR_ANY (for '`inet`') or to `::` for IPv6.

3.1.1 Instantiating a previously created interface

The `create-if` will not restore any previously configured attributes of the interface (e.g., IP address) that exist in the persistent store. To resurrect the information in the persistent store, the `up-if` command may be used. The syntax of `up-if` is symmetric to the `create-if` command:

```
# ipadm up-if [-f {inet, inet6}]  <interface>
```

3.2 Destroying the physical interface

```
# ipadm delete-if [-f {inet, inet6}]> <interface>
```

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

By default, this command flushes any references to the interface from the peristent store so that the interface will not recreated on reboot.

3.2.1 Deinstantiating the physical interface

The `down-if` command, whose syntax is similar to the `delete-if` command, will dismantle the physical interface from the running-config without removing the information pertaining to the interface from the persistent store.

3.3 Modifying properties of the interface

The commands

```
# ipadm set-ifprop [-t] -p <name>=<value>[,<value>,..] [-p ..] <interface>
# ipadm show-ifprop [-P] [-p <name>,<name>,..] [<interface>]
# ipadm reset-ifprop [-p <name>,<name>,..] <interface>
```

may be used to modify or view per interface properties. The `reset-ifprop` command is used to reset the given properties to the values had at the startup. If no properties are specified, all properties are reset. Also see Section 11 for a discussion on the relation with global properties, and Section 12.2 for additional interface properties specific to IPMP.

The list of tunables that will be made available through `ipadm` may be found at [4].

3.4 Examples

The interface status can be viewed through the command

```
# ipadm show-if
```

Sample output is shown below[2]:

INTF	LINKS	MTU	STATE	FLAGS
lo0	–	8232	ok	MULTICAST,VIRTUAL
e1000g0	e1000g0	1500	ok	BROADCAST,MULTICAST
e1000g1	e1000g1	1500	down	BROADCAST,MULTICAST,ROUTER
ipmp0	qfe0 qfe1	1500	failed	BROADCAST,MULTICAST,IPMP
qfe0	qfe0	1500	down	BROADCAST,MULTICAST
tun0	–	1480	down	MULTICAST,P2P,NOARP
vni0	–	68	ok	NOARP,NOXMIT,VIRTUAL
bge0	bge0	9000	ok	MULTICAST,NORTEXCH,NONUD

Managing interface properties:

```
# ipadm set-ifprop -p mtu=900 link0          # sets the 'mtu' to 900 for interface 'link0'
# ipadm set-ifprop -p metric=2 link0         # set the 'metric' to 2 for interface 'link0'
# ipadm set-ifprop -p reasm_timeout=60 link0 # sets the reassembly fragment timeout
                                               on link0 to 60 seconds

# ipadm show-ifprop
```

INTF	PROPERTY	PERMS	VALUE	DEFAULT	POSSIBLE
intf0	mtu	rw	784	1500	68-9000
intf0	metric	rw	2	1	–
intf0	reasm_timeout	rw	60	15	1 - 6000

4 Static address management

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

4.1 Add static address

```
# ipadm create-addr [-t] [-d <dstaddr>] -i <interface> <address>
```

Configures `<address>` as the local IPv4 or IPv6 address on `<interface>`. The `<address>` should be a numeric address in CIDR notation.

The `-d dstaddr` specifies the destination IP address of a point-to-point interface.

The `<interface>` 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`) or by using Stateless/Stateful Address configuration, using `create-ipv6addrs` (see Section 5).

This command is persistent by default, i.e., an address added by '`ipadm create-addr <...>`' updates the persistent store. The '`-t`' flag can be used for temporary changes. Attempting to persistently set an address on an interface that has been created with the '`-t`' option will generate an error. Attempting to temporarily set an address (i.e, with the `-t`) on an interface that has been created persistently will generate a warning, and the `create-addr` information will not be used to update the persistent store..

[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

4.2 Deleting a static address on interface

The command

```
# ipadm delete-addr [-t] -i <interface> <address>
```

removes the `<address>` that has been previously configured as the local address on an existing `<interface>` by using the `ipadm create-addr` command (See Section 4.1). Argument semantics are the same as those described in Section 4.1

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 8

5 IPv6 Stateless/Stateful address management

Groups of addresses defined by IPv6 Stateless/Stateful address configuration are managed by the `create-ipv6addrs` and `delete-ipv6addrs` commands. These groups of addresses are collectively referenced by a user-supplied `label` that can be any user-defined string that is unique on the system. The `<label>` must be supplied with the `create-` command, and must be used as the object of subsequent `delete-` commands. Labels are unique per interface.

5.1 Creating IPv6 addresses using stateless/stateful addrconf

The command

```
# ipadm create-ipv6addrs [-t] [-T IPv6 Interface ID] \
                [--stateless={true,false}] [--dhcpv6={true,false}] \
                -i <interface> <label>
```

The `IPv6 Interface ID` is the token to be used for generating auto-configured addresses. If not specified, the default Interface ID is generated from the MAC address of the 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.

When `stateless` is set to true, IPv6 addresses will be autoconfigured based on prefixes advertised by routers as described in RFC 2462. When set to false, no autoconfiguration based on advertised prefixes is performed. By default, this option is set to true.

When `dhcpv6` is set to true, 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. This option is enabled by default.

The default IAID for DHCPv6 is the ifIndex number of `interface` and is kept in stable storage to ensure consistency of the (interface name, IAID) pair. If this default is unavailable, arbitrary number will be generated. Future extensions to the `create-ipv6addrs` 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-ipv6addr` command will allow a custom DUID to be supplied using the `-C` option.

5.2 Deleting IPv6 addresses created using stateless/stateful addrconf

The command

```
# ipadm delete-ipv6addrs [-t] [-release] -i <interface> <label>
```

will delete the group of addresses associated with `label` on `interface`. If these addresses are obtained by Stateful Address Configuration, the default behavior for deletion will be done without notifying the DHCP server and the current lease is recorded for later use. With `-release`, the IP addresses are relinquished by notifying the server.

The `show-ipv6addrs` command can be used to view the addresses associated with a `label` on a given interface. The command `show-dhcpv6` shows the status of the IPv6 addresses created by DHCPv6 for the given label.

5.3 Examples

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

```
  # ipadm create-ipv6addrs -T ::abcd -d true -s true -i link0 myv6addrs
  # ipadm show-ipv6addrs -x -i link0 myv6addrs
LABEL        PFXORIGIN    STATE      FLAGS  DHCPv6  STATELESS ADDR/MASK
myv6addrs    wellknown    preferred  ---    yes     yes       fe80::abcd
             routeradv    preferred  ---    yes     yes       2002:a08:39f0:1:203:baff::abcd/64
```

The field `PFXORIGIN` describes the origin of the Prefix (see RFC 4293, Page 15). For a description of `FLAGS` and `STATE`, refer to Section 7.

A labeled group of IPv6 addresses created using `create-ipv6addrs` may be deleted by:

```
# ipadm delete-ipv6addrs -i link0 myv6addrs
```

6 IPv4 DHCP address management

As with IPv6 stateless/stateful addresses, IPv4 DHCP addresses are managed as labeled groups. The address(es) are managed by the `create-dhcp` and `delete-dhcp` commands. The DHCP address can be referenced by a user-supplied `label` that can be any user-defined string that is unique for the interface on the system. The `<label>` must be supplied with the `create-` command, and must be used as the object of subsequent `delete-` commands.

6.1 Creating IPv4 addresses using DHCP

The command

```
# ipadm create-dhcp [-t] -i <interface> <label>
```

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.

6.2 Deleting DHCP addresses

The command

```
# ipadm delete-dhcp [-release]  -i <interface> <label>
```

will delete the address associated with `label`. By default, the deletion will be done without notifying the DHCP server and the current lease is recorded for later use. With `-release`, the IP addresses are relinquished by notifying the server.

The `show-dhcp` command can be used to view the address associated with a `label`.

Address state as printed by ipadm	Address state flag returned by ipadm_show_address	Semantics
preferred	IFA_PREFERRED	Default value in the absence of other qualifiers
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 15)
tentative	IFA_TENTATIVE	DAD has been initiated
duplicate	IFA_DUPLICATE	DAD failure
optimistic	IFA_OPTIMISTIC	Optimistic DAD

Table 1: STATE information returned by ipadm show-address

6.3 Example

The example below shows an address obtained by DHCP.

```
 # ipadm create-dhcp -i link0 myv4addr
 # ipadm show-dhcp -i link0 myv4addr
LABEL         ADDR/MASK   STATE   BEGIN            EXPIRE
myv4addr      10.2.3.8/24 BOUND   04/16/2009,13:50 04/16/2009,15:00
```

7 Displaying IP address on the system

The `/sbin/ipadm` invocation:

```
# ipadm show-addr <interface>
```

will display all the IP addresses available on the system. The information will contain the following fields:

- INTF, the name of the IP interface on which the address is configured.
- ORIGIN, which desribes the origin of the address (see RFC 4293, Page 14). The fields printed by `/sbin/ipadm` will correspond to the supported sub-commands, namely `{ipv6addr, static, dhcp}`. Here, `ipv6addr` indicates that the address was obtained by Stateless/Stateful Address Configuration, and the `show-ipv6addr` should be used for a detailed view of the IPv6 Addresses. The `dhcp` indicates that the IPv4 address was configured using DHCP, and the `show-dhcp` command may be used for further details.
- STATE, which descibes the Address status and may be one of the states desribed in Table 1

7.1 Examples

```
# ipadm create-addr -i tun0 -d 12.1.2.2 192.168.15.190
# ipadm create-addr -i vni0  11.2.3.1/8
# ipadm show-addr
```

INTF	ORIGIN	STATE	ZONES	FLAGS	ADDR/MASK
lo0	–	–	global,zone1	---	127.0.0.1/8
e1000g1	static	duplicate	zone1	---	193.168.16.190/24
e1000g1	ipv6addrs	deprecated	global	---	2002:a08:39f0:1:203:baff::abcd/64
e1000g1	ipv6addrs	invalid	zone1	-t-	2002:a08:3980:1:219::4764:8349/64
ipmp0	dhcp	tentative	global	---	192.168.16.8/24
tun0	static	preferred	global	U-p	192.168.15.190/32->12.1.2.2/32
vni0	static	invalid	global	--p	11.2.3.1/8

```
U: Unnumbered    p: Private  t: Temporary
```

8 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 Soalaris 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_show_address()` 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_show_address()` will also report ORIGIN and STATE as described in Section 7. The interface name returned by the library call will be the logical interface name.

9 Refreshing IP Addresses

A common command

```
/sbin/ipadm refresh-addr -i <interface> <<addr>|<label>>
```

is provided to re-verify the IP address(es). If `<addr>` is specified, then DAD will be restarted (if necessary) on the static `<addr>`. If `<label>` is specified, the prefix used for configuring the address(es) implied by `<label>` will be revalidated to verify and extend its lifetime.

10 Address Properties

The `ipadm set-addrprop` and `show-addrprop` commands may be used to tune Interface Address Properties. The syntax of the commands are:

```
# ipadm set-addrprop -p <name>=<value>[,<value>,..] [-p ..] -i <interface> <<address>|<label>>
# ipadm show-addr prop [-p <name>,<name>..]  -i <interface> <<address>|<label>>
```

Supported properties will include:

- source-address selection preference: `src-addr-sel = {deprecated, nolocal}`

- boolean property `private = {true, false}`. If set to true, the address is not advertised via Routing Protocols. Default: `false`

- changing the netmask `netmask = <len>` sets the netmask length of the address indicated.

- changing the zone associated with address: `zone = <zonename>`. Note that this command should only be used for Shared IP stack zones (`dladm` should be used for exclusive zones).
- boolean property `enabled = {true, false}`. If set to true, the address is a valid address on the system, and may be used as the source/destination of packets from/to the system.

The current values for the properties can also be obtained by doing `ipadm get-addrprop`.

11 Global Property/tunable management

The commands

```
# ipadm set-prop [-t] -p <name>=<value>[,<value>,..] [-p ..] <protocol module>
# ipadm show-prop [-P] [-p <name>,<name>..] [<protocol module>]
# ipadm reset-prop [-p <name>=<value>[,<value>,..] [-p ..]] <protocol module>
```

provide TCP/IP tunable managment equivalent to the features of the `/sbin/ndd` command to be applied to the appropriate `<protocol module>` (i.e., tcp, ip, udp etc.) for all applicable interfaces. Details of each of these sub-commands are provided in the sections below. The commands themselves allow for setting/viewing property values that are global to the protocol module itself. Note that this global value may be over-ridden by a per-interface value for the property using the `set-ifprop` command described in Section 3.3

11.1 Legacy support of ndd tunables

The `ipadm/libipadm` interfaces have the facility to set tunables per interface. This presents an issue when the same property also has a tunable that is supported by `ndd`. For example, a property `ip_icmp_err_rate` may be tuned via `ipadm` to have different values on `intf0` and `intf1`, but `ndd` does not have the facility to print more than one (global) value. In such a case, the value that `ndd` can set/get will be defined as the value interpreted in the `set-prop` and `show-prop` commands, and the one that would be configured for the tunable unless it is over-ridden by a per-interface specific value set via `ipadm set-ifprop`.

11.2 Setting properties

The command
'`ipadm set-prop [-t] -p  <name>=<value> [,<value>,..] [-p ..] <protocol module>`'
applies the name-value pair property setting

11.3 Resetting properties

The command
'`ipadm reset-prop [-p  <name> [-p ..]] <protocol module>`'
resets the specified properties to the values they had at the startup.

11.4 Getting properties

The '`ipadm show-prop [-P] [-o field] [-p <name>,..] [<protocol module>]` '. Sample output is available in Section 11.5.

11.5 Use cases and examples

The `-p`, and `interface` selectors may be used to pick out specific sets of rows from the output above.

```
# ipadm set-prop -p ttl=64 ip     # sets the 'ttl' to 64 on 'ip' module.
# 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        PERMS      VALUE   DEFAULT   POSSIBLE
 ip     forwarding      rw         yes     yes       yes/no
 ip     ttl             rw         64      255       1-255
 ip     reasm_timeout   rw         60      15        1-6000
  :
```

Some use cases are described below:

- 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

```
# ipadm create-ipmp [-t] [-g <groupname>] [-i <under-interface>...] <ipmp-interface>
```

Creates an IPMP interface with an optional `groupname`. Interfaces can be optionally added to the IPMP group during the creation of the interface. If no groupname is specified, the name will be set to the string identified by `ipmp-interface`. This command creates both IPv4 and IPv6 IPMP interfaces.

This command is persistent by default, i.e., an IPMP interface created by '`ipadm create-ipmp <...>`' will be recreated on reboot. The '`-t`' option may be used for temporary interface creation.

```
# ipadm delete-ipmp [-t] <ipmp-interface>
```

Deletes the IPMP group identified by `ipmp-interface`. It deletes both IPv4 and IPv6 interfaces. This command will generate an error if the IPMP group is not empty.

```
# ipadm add-ipmp [-t] -i <under-interface> [-i <under-interface>...] <ipmp-interface>
```

Adds one or more underlying interfaces to the IPMP group `ipmp-interface`.

```
# ipadm remove-ipmp [-t] -i <under-interface> [-i <under-interface>...] <ipmp-interface>
```

Removes one or more underlying interfaces from the IPMP group `ipmp-interface`.

```
# ipadm show-ipmp <ipmp-interface>
```

Shows information about the IPMP group `ipmp-interface`.
Sample output:

INTF	UNDERINTF	STATE	ACTIVE	STANDBY	TESTADDR
ipmp0	net0	ok	yes	no	11.1.2.3
	net1	ok	no	yes	11.1.2.4

12.1 Addresses on IPMP groups Properties and address creation

Data and test addresses can be created statically or by dhcp using ipadm address creation commands described in section 4.1, section 6 and section 5.

12.2 IPMP group properties

Properties of ipmp interfaces and underlying interfaces can be set using `ipadm set-intfprop <...>` command described in section 3.3. In addition to the interface properties described in Section 3.3 the following properties specific to IPMP will be supported:

- `standby = {true, false}`. This applies only for underlying interfaces and an error will be generated if set on an IPMP interface. Default: `false`.
- `groupname = <string>`. For an IPMP interface, this specifies the name of the group, while for an underlying interface, this specifies the group it is part of. This property cannot be changed for an underlying interface, if the interface is already part of an IPMP group.

12.3 Example

```
# Create group interface 'grp0' with groupname 'testgrp0'
ipadm create-ipmp -g testgrp0 grp0

# Add interfaces net0 and net1 to the group
ipadm add-ipmp -i net0 -i net1 grp0

# Add data addresses
ipadm create-addr -i grp0 10.1.2.2
ipadm create-addr -i grp0 10.1.2.3

# Add test addresses
ipadm create-addr -i net0 11.1.2.3
ipadm create-addr -i net1 11.1.2.4

# Make net1 a standby interface
ipadm set-intfprop -p standby=true net1

# Show grp0 group details
ipadm show-ipmp grp0
```

INTF	UNDERINTF	STATE	ACTIVE	STANDBY	TESTADDR
grp0	net0	ok	yes	no	11.1.2.3
	net1	ok	no	yes	11.1.2.4

```
# Show address details of grp0
ipadm show-addr
```

INTF	ORIGIN	STATE	FLAGS	ADDR/MASK
...				
grp0	static	preferred	---	10.1.2.2/24
grp0	static	preferred	---	10.1.2.3/24
...				

```
# Remove interface net1 from the group
ipadm remove-ipmp -i net1 grp0

# Deleting group grp0
ipadm remove-ipmp -i net0 grp0
ipadm delete-ipmp grp0
```

13 System call details for property management

`ndd(1m)` uses Mentat's ND_SET and ND_GET system calls to extract tunable 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. Minimally, two interfaces would be required: one ioctl to set properties, and one to get properties. The `SIOCSMODPROP`[3] system call would take[4] a name-value setting that is passed down to the kernel. The `SIOCGMODPROP` ioctl would take a name, and return a parsed value. The format of the data passed to/from the kernel could loosely follow the format of the `dld_ioc_mac_prop_t` used in Brussels I. We have prototyped the ioctl

[3] name subject to change

[4] among other things

structure `mod_ioc_prop_t` to be

```
typedef struct mod_ioc_prop_s {
    int mpr_version;
    uint_t mpr_flags;
    char mpr_ifname[LIFNAMSIZ];
    uint_t mpr_perm_flags;
    char mpr_name[MAXPROPNAMELEN];
    uint_t mpr_valsize;
    char mpr_val[1];
} mod_ioc_prop_t;
```

Note that this is a work in progress and more information will be added to this structure, so that we could retrieve multiple pieces of information (current value, default value and all possible values) in a single `get` invocation.

The existing ND_SET and ND_GET ioctls would be declared Obsolete and support for these ioctls would be removed over time, after sufficient time has elapsed for third party applications to switch to the MODIOC version.

14 Mechanism for restoring Persistent information

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 [3].

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

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-intf`.

14.1 Privileges model associated with the data store

The initial implementation of the repository for tracking persistent information will use the flat-file model, saving configuration in a Consolidation Private file following the conventions of the datalink configuration file used for PSARC 2006/499. Specifically, ipadm configuration information will be stored in `/etc/ipadm/ipadm.conf`, and the format of data in the file will be Private to `libipadm` and therefore, not modifiable by the Administrator. The `ipadm.conf` file will be owned by the `netadm` user-id (also used for NWAM configuration). Thus any tool that wishes to write to the repository will need to have the '`file_dac_write`' privilege through `exec_attr`.

Note that this model has a drawback, in that any user-application that links to `libipadm` will need the `file_dac_write` privilege if it directly/indirectly makes persistent IP configuration changes. A better solution would be to use pre-defined interfaces with `svc.configd(1M)`, and this ideal will be achieved after ongoing discussions converge on an SMF service model that achieves the needs of several related projects in this area [2]

14.2 Transitioning from existing Persistence restoration mechanisms

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:

- New installations should only use the `ipadm` interface. Thus the `net-physical` script will first query the `ipadm` persistent store for known interfaces using the `ipadm show-intf` command, and use (preferring, when multiple conflicting configuration methods are encountered,) the `ipadm` configuration to reconstruct IP interfaces on reboot.

  In the interim period when all `ifconfig` operations are not supported in `ipadm`, the `net-physical` script will merge the configuration requested with precedence given to `ipadm.conf`

- When older methods such as `/etc/hostname.<..>` files are detected, and no ipadm persistent configuration is found for the interface, `net-physical` will fall back to legacy methods. However, a subsequent (to be added and enhanced, as `ipadm` acquires compatibility with `ifconfig` options) smf service will update this configuration information to input in the `ipadm` store for the next reboot.

15 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 -i net0 10.1.2.3/24
```

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

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-ipv6addrs` 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 labels, each corresponding to the Interface ID specified in the `create-ipv6addrs` command.

16 Acknowledgments

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

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

References

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

[2] "libipadm data store access"
. http://mail.opensolaris.org/pipermail/brussels-dev/2009-March/001289.html.

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

[4] "Summary of select ndd(1m) tunables in TCP/IP supported by ipadm"
. http://cr.opensolaris.org/~girishmg/ipadm_prop.htm.

[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.