Brussels II - ipadm and libipadm Design Specification

Girish Moodalbail
Vasumathi Sundaram
Sowmini Varadhan

Project Brussels I-Team
brussels-dev@opensolaris.org

Solaris Networking
Sun Microsystems, Inc.

Revision 1.15
March 02, 2010

Contents

Tables

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 Stable Interface for Boot-time NDD Configuration
- 6597312 ip addrs per if not respected during boot; limited to 256 addresses per if
- 4404812 set network card configuration with ndd-conf.sh

Essentially TCP/IP tunables are modified only via *ndd(1M)* today. The value range that can be applied on a tunable is not clearly known to the user. They have to skim through the source code to find the value range that can be applied. Further these settings are not automatically applied on reboot. Prior to *smf(5)*, customers were able to (merely) simulate persistence by editing the */etc/rc.d* scripts, but after the introduction of *smf(5)*, with the interdependency of *smf* services the start-up sequence predictability are much subtler than *rc.d* scripts and so it's not clear when the tunables 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.so*' that will

- export interfaces that can be utilized by other programs, including */usr/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 TCP/IP configuration library in *libipadm.so* that can be used by multiple configuration tools. The existing commands (*ifconfig*, *dladm*, *etc.*) are 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 and will become a thorn in the side of future development and maintenance, since the code is Private and subject to change. The latter approach is inefficient.

In contrast, *libipadm.so* 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.so* 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*, that satisfy the following requirements:

- *libipadm.so* 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) 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:

  - IP interface creation (*ifconfig plumb)*/ deletion (*ifconfig unplumb)*
  - address management (add/delete/show)
  - tunable management (set/get).

  The proposed syntax of these commands are described in Section 4, 5, 6 & 7.

- *libipadm.so* should have sufficient 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 targeted 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 7.

- 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. See CR 4616660 for more information on 'status tunables' that were moved to MDB. Note: This part has been already delivered into Nevada (snv_112).

- ndd compatibility: */sbin/ndd* will be converted to use the interfaces provided by *libipadm.so*, and the implementation of these interfaces may themselves have Consolidation Private stability. However, since there may be applications (for e.g., routing daemons like Quagga) 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[1], and followed in */sbin/dladm* and */sbin/flowadm*.

- there will be a 1-1 mapping between an *ipadm* invocation and a library interface. In addition, all commands will follow the existing conventions used in */sbin/dladm* and */sbin/flowadm* for parsable output, column selection.

- *libipadm.so* will supersede the Consolidation Private interfaces provided by *libinetcfg*, and *libinetcfg* invocations in ON will be converted to use *libipadm.so*.

- Many, many scripts slog their way through ifconfig output today, and those will be simplified considerably by ipadm through committed and easy-to-parse output.

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

2 High-Level Architectural Overview

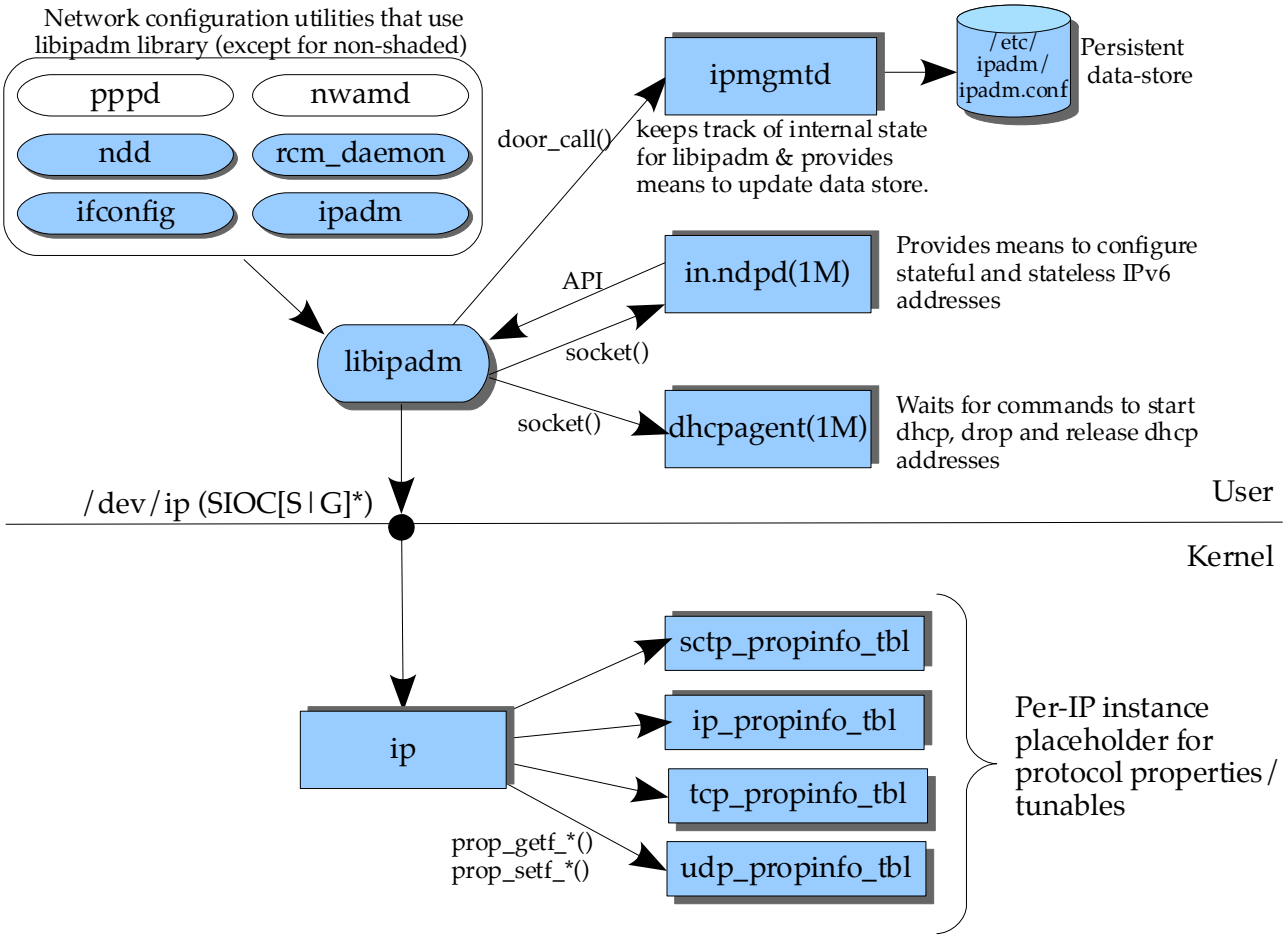

Figure 1: High-level relationship between various system components involved in Brussels II design.

The *ipadm(1M)* command will be used to create IP interfaces and configure IP addresses on them. It will also be used to set/get/reset properties on interface, address and protocol. It will do this by first opening a *libipadm.so* library handle and with this handle, calls into API's implemented in the *libipadm.so* library. The handle is an opaque data structure and contains socket file descriptors and a door descriptor to *ipmgmtd* daemon, among other things. This library will be responsible for interacting with *ip* kernel module using a set of socket ioctls (SIOC[S|G]*), to create IP interfaces and addresses and to set/get protocol properties.

The daemon, for persistent operations, updates the datastore with the IP interface/address configuration or TCP/IP property value changes, so that it can be re-applied on system reboot. The daemon also maintains a list of address objects (Section 10.1), with every node in the list mapping address object name to logical interface on which the address is configured.

The protocol properties managed by ipadm are per-IP instance. Inside the kernel they are captured in an array of *struct mod_prop_info_s* and this array hangs off respective protocol stack instance structure. Each property has both get and set callback function pointer defined in the *mod_prop_info_s* structure which gets or sets the value for that property, respectively.

While configuring DHCP addresses on an interface, *libipadm.so* interacts with *dhcpagent(1M)* using *libdhcpagent (*which uses a loopback socket based IPC) and issues commands, such as, *DHCP_START, DHCP_DROP, DHCP_RELEASE & DHCP_EXTEND* to start dhcp, drop dhcp address, release dhcp address and extend a lease of a dhcp address respectively.

While configuring IPv6 stateful and stateless address configuration, *libipadm.so* interacts with *in.ndpd(1M)* daemon to configure auto-configuration addresses on the given interface. We have modified *in.ndpd(1M)* to listen, using AF_UNIX sockets, for messages *NDPD_DISABLE_ADDRCONF, NDPD_CREATE_ADDRS* and *NDPD_DELETE_ADDRS:*

- The *NDPD_DISABLE_ADDRCONF* message instructs *in.ndpd* to disable auto-configuration for a given interface (i.e., no stateless or stateful IPv6 addresses will be generated)
- The *NDPD_CREATE_ADDRS* message instructs *in.ndpd* to start the Router Solicitations and configure IPv6 addresses using the prefixes received in the Router Advertisement.
- The *NDPD_DELETE_ADDRS* message instructs *in.ndpd* to remove the auto-configured addresses and disable auto-configuration on the given IPv6 interface.

See Section 5.1.1 and 5.1.2 for more information.

Other than ipadm, the other consumers of libipadm are */usr/sbin/ndd(1M), /sbin/ifconfig(1M)* and *rcm_daemon*. Other future potential consumers are *nwamd(1M)* and *pppd(1M).*

3 ipadm(1M) CLI

The *ipadm(1M)* command is used to configure IP interfaces, IP addresses, modify interface properties and protocol properties.

The next few sections will describe every ipadm subcommand in great detail and with examples. Section 4 covers Interface management (interface creation, deletion and modifying interface properties). Section 5 covers Address Management (address creation, deletion, marking them up/down, refreshing and modifying address properties). Section 6 covers set of subcommands that manages temporary deletion of objects and then restoring them back. Section 7 covers managing protocol properties. Section 8 covers *libipadm.so* API's that are invoked for every *ipadm* subcommand.

All of the show-* subcommands have an option (-p for *show-addr* and *show-if*; -c for *show-ifprop, show-addrprop* and *show-prop*) that displays the output in a machine-parsable format. The output format is one or more lines of colon (:) delimited fields. The fields displayed are specific to the subcommand used and includes only those fields requested by means of the -o option, in the order requested.

In addition, the terms “property”, “tunable” and “parameter” are used interchangeably throughout the document. The terms, “running configuration” and “active configuration” are used interchangeably. The terms, “saved configuration” and “persistent configuration” are used interchangeably.

Further, the definition of -t option, used in all the *ipadm* subcommands, is that it only applies to the running configuration and is valid as long as the object is enabled.

4 Interface management

This section describes a set of ipadm subcommands that are used to configure IP interfaces.

4.1 Creating an IP interface

 # ipadm create-if [-t] <interface>

The *create-if* subcommand creates an IP interface object, which can later be displayed with *ipadm show-if* and used with *ipadm create-addr* to create address objects on that interface. In the current implementation, *create-if* performs an *ifconfig plumb* for both IPv4 and IPv6. The address on the IPv4 interface will be set to INADDR_ANY (0.0.0.0) and the address on the IPv6 interface will be set to unspecified address "::". This subcommand does not need to be explicitly invoked in the typical case (e.g., prior to *ipadm create-addr*, as *create-addr* implicitly creates an interface if it is not already created).

Today in Solaris Nevada, whenever an IPv6 interface is plumbed, the kernel always generates a link-local address. In our case, we want to create an 'unspecified address, ::', therefore we are introducing a Consolidation Private interface flag called IFF_NOLINKLOCAL. When this flag is set the kernel will not generate a link-local address by default.

By default, this subcommand persists the information so that on the next reboot this interface will be instantiated. The '-t' option, when used, changes running configuration alone. An interface is *enabled* for IPv4 and IPv6, implicitly, when it is created. See Section 6, for how to use *disable-if/enable-if* subcommands to disable and enable an interface.

4.2 Deleting an IP interface

 # ipadm delete-if <interface>

This subcommand deletes the interface object from running configuration, that is, it performs the *ifconfig unplumb* for both IPv4 and IPv6. All the addresses configured on that interface will be torn down. This subcommand flushes interface properties and address objects configured on that interface from the persistent store so that, on reboot, the interface and the address objects on that interface will not be instantiated. See Section 6, for how to disable an IP interface from running configuration using *disable-if* subcommand.

4.3 Displaying IP interfaces

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

This subcommand displays IP interface configuration, for the given *interface* from the active configuration and the persistent configuration. If the *interface* is not provided then the information is displayed for all the interfaces on the system, including the ones that are only in the persistent store. For each *interface*, following fields are displayed:

STATE	Semantics
offline	The interface is offline (i.e. IFF_OFFLINE is set) and thus the interface cannot send or receive traffic. See *if_mpadm(1M).*
failed	Indicates that the datalink is down (i.e. IFF_RUNNING is not set). If the interface is part of an IPMP group it could also mean that the interface has failed (i.e., IFF_FAILED is set). See *in.mpathd(1M).*
down	Indicates that the interface is administratively down, preventing any IP packets from being sent or received through it (i.e. when ALL the addresses on this interface do not have IFF_UP set)
ok	Indicates that the required resources for an interface are allocated. For some interfaces this also indicates that the link is up.
disabled	Indicates that the interface has been disabled from the active configuration using 'disable-if' subcommand.

Table 1: Interface STATE information as displayed by ipadm show-if

- IFNAME: The name of the IP interface.
- STATE: Describes the interface status and may be one of the states described in the Table 1 below
- CURRENT: For interface objects in active configuration, it indicates any of the following.

Flags	Semantics
b	interface supports broadcast
m	interface supports multicast
p	interface is a point-to-point link
v	virtual interface (for e.g., vni(7d), loopback), the physical interface has no underlying hardware.
I	IPMP meta interface
s	IPMP interface is marked standby administratively. See in.mpathd(1M)
i	IPMP interface is inactive. See in.mpathd(1M)
V	interface is a VRRP interface[2]
a	VRRP interface is in accept mode (~IFF_NOACCEPT)
4	`interface can handle IPv4 packets`
6	interface can handle IPv6 packets
Note: b and p are mutually exclusive.	

Table 2: Interface FLAGS information as displayed by ipadm show-if

- PERSISTENT: Specifies the configuration that will be applied when the interface object is instantiated on reboot or re-enabled using 'enable-if' subcommand. It can be any or all of s, 4 & 6 (see above).

NOTE: Though we are concealing the kernel implementation details (v4 and v6 *ill's*) from ipadm, by plumbing both v4 and v6 interface when the user invokes *create-if,* in reality users can still use *ifconfig* to unplumb just one *ill*. So the CURRENT column will not always have both '4' and '6'.

2 http://arc.opensolaris.org/caselog/PSARC/2009/388/

Examples:

Display all the IP interfaces from active and persistent configuration. Note that *e1000g1* and *tun0* IP interfaces are only in persistent store and not in running config. While *e1000g1* will support both protocols after it's re-enabled, *tun0* will support just ipv6 after it's re-enabled.

 # ipadm show-if

IFNAME	STATE	CURRENT	PERSISTENT
lo0	ok	-m-v-----46	---
e1000g0	ok	bm-------4-	-46
e1000g1	disabled	----------	-46
ipmp0	failed	bm--I----46	-46
bge0	failed	bm---s---46	-46
tun0	disabled	----------	--6
vni0	--	---v-----46	-46

4.4 Modifying properties of an interface

 # ipadm set-ifprop [-t] -m protocol -p <prop>=<value[,..]> <interface>
 # ipadm reset-ifprop [-t] -m protocol -p <prop> <interface>

The *set-ifprop* subcommand is used to modify an interface property to the value specified by the user. If the property takes multiple values then the values should be specified with comma as the delimiter. If the operation is successful, the value will be persisted, unless -t option is specified. The persisted value is reapplied on reboot.

The *reset-ifprop* subcommand is used to reset an interface property to it's default value. If the operation is successful the persisted value will be purged from the saved configuration, unless -t option is specified. Only one property can be modified at a time.

Note that one cannot perform persistent operation on temporary object, that is, if the interface is temporarily created, then one cannot apply the interface property persistently. If the user does so, then an error will be returned. Only one property can be modified at a time.

The -m option can be a transport protocol like *tcp* or *udp*, or it can be *ip*, *ipv4* or *ipv6* (that is some properties are common to IP and some are specific to a version of IP). For example: Some interface properties, such as *mtu*, can be set to different values for IPv4 or IPv6. Further they have different *minimum* values (for IPv4 it's 68 and for IPv6 it's 1280), which adds to the confusion. So, to make things less confusing and to disambiguate the value being set, -m option is mandatory.

Note that, as with the "*dladm set-linkprop*" command, the interface argument is mandatory, since the *ipadm set-ifprop* is a potentially destructive command to apply on all interfaces thus requiring a mandatory interface to prevent costly accidents.

Section 7 describes a method for setting tunables on protocols, on global basis. Some interface properties such as IP *forwarding* has a global setting, that apply to all IP interfaces created on that IP instance. In such cases, if the value of the interface property for some interface is different from the global setting, the per-interface value would be applied for that interface. Note that not all of the properties that apply to the IP protocol have per-interface tunables. For example tunables related to Strong and Weak end-systems (CR 4173841) do not have per-interface settings, and may not be settable with *set-ifprop*.

Following are the list of per-interface properties that will be supported in the first phase: *arp, mtu, metric, nud, usesrc, exchange_routes* and *forwarding*. For definitions of these properties please refer

to *ipadm(1M)* man page[3]

 # ipadm show-ifprop [-m protocol] [[-c] -o <field>[,...]] [-p <prop>,...] [<interface>]

This subcommand displays the current and persistent value for a given interface property, among other things. Several properties of interest can be retrieved at a time by providing comma separated property names to -p option.

Providing an IP interface name is optional. When it is not provided, for all the IP interfaces on the system, including the IP interfaces in persistent store, the interface properties will be displayed. Similarly providing property list is optional too, when not provided all the interface properties for a given or all interfaces is displayed. For each property, following fields are displayed in the output:

- IFNAME - Name of the IP interface
- PROPERTY - Name of the property
- PROTO - Specifies the protocol to which the property belongs
- PERM - specifies if the value is read only, write only or read and write
- CURRENT - The current value of the property
- PERSISTENT - The persistent value of the property
- DEFAULT - Default value of the property
- POSSIBLE - Possible range of values for a given property

Examples:

Changing IPv4 *mtu* value, temporarily, for net0 to 1100

 # ipadm set-ifprop -t -m ipv4 -p mtu=1100 net0

Disabling Neighbor unreachability discovery (*nud)* on net0

#ipadm set-ifprop -m ipv6 nud=off net0

Enabling IPv4 *forwarding* and IPv6 *forwarding* on net0

 # ipadm set-ifprop -m ipv4 -p forwarding=on net0
 # ipadm set-ifprop -m ipv6 -p forwarding=on net0

Displaying the IPv4 *mtu,* IPv4 *nud* and IPv4 and IPv6 *forwarding* for net0

 # ipadm show-ifprop -m ipv4 -p mtu net0

IFNAME	PROPERTY	PROTO	PERM	CURRENT	PERSISTENT	DEFAULT	POSSIBLE
net0	mtu	ipv4	rw	1100	1500	1500	68-1500

 # ipadm show-ifprop -m ipv6 -p nud net0

IFNAME	PROPERTY	PROTO	PERM	CURRENT	PERSISTENT	DEFAULT	POSSIBLE
net0	nud	ipv6	rw	off	off	on	on,off

 # ipadm show-ifprop -p forwarding net0

IFNAME	PROPERTY	PROTO	PERM	CURRENT	PERSISTENT	DEFAULT	POSSIBLE
net0	forwarding	ipv4	rw	on	on	off	on,off
net0	forwarding	ipv6	rw	on	on	off	on,off

3 http://zhadum.east/export/ws/ipadm/commitment/ipadm.1m.txt

4.4.1 routeadm(1M) and forwarding property

The *routeadm(1M)* command is used to administer system-wide configuration for IP forwarding and routing. It currently uses */sbin/ndd* to enable IP forwarding. Since *ndd(1M)* does not provide persistence, the persistence is provided using a SMF service. The whole purpose of this service, when enabled, is to execute *ndd -set /dev/ip ip_forwarding 1* on reboot. Further *routeadm(1M)* does not support enabling/disabling per-interface forwarding.

The *ipadm(1M)* command, which provides persistence, allows setting of forwarding globally (*set-prop)* or per-interface (*set-ifprop)* seems like an obvious fit for this feature. So this project will mark r*outeadm(1M)* interfaces to enable/disable IPv4/IPv6 forwarding "*Obsolete"* and will eventually EOF that feature. Until that point in time, when both the tools continue to exist, we will modify the SMF script *(/lib/svc/method/svc-forwarding)* to invoke *ipadm*. So that both *routeadm(1M)* and *ipadm(1M)* will be modifying the same repository and hence will have the same view.

5 Address Management

This section describes set of ipadm subcommands that are used to create IP addresses, delete them, bring up/down them, modify properties on them, and finally, display them.

Note that one cannot perform persistent operation on temporary object, that is, if the interface is temporarily created, then one cannot create the addresses persistently on that interface. Similarly, if the address object was created temporarily, then one cannot perform *up-addr, down-addr* and modify address properties persistently. If the user does so, an error will be returned.

5.1 Address Creation

This section describes ipadm subcommands that manage static IPv4 and IPv6, DHCP and IPv6 ADDRCONF addresses on interfaces. The general synopsis of the i*padm create-addr* subcommand is:

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

Each of the types, *static*, *dhcp* and *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:

- 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* ensures that *addrobj* values identify a unique address or set of addresses on the system (in the case of IPv6 stateful/stateless auto-configuration). 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.

In all the *create-addr* invocation if the underlying interface on which the address is created is not created then the interface will be created and then the address will be created. If the address creation was persistent then the implicit interface creation is also persistent and if the address creation was temporary then the implicit interface creation is also temporary.

Further, all the address objects are enabled when they are created. See Section 6 on how to use *disable-addr/enable-addr* subcommands to disable/enable address objects.

5.1.1 Adding static address

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

This subcommand 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*.
2. a hostname whose numeric value is uniquely obtained using the resolver order specified for *hosts* or *ipnodes* in *nsswitch.conf*. If the hostname entry is not found in */etc/hosts* or if there are multiple entries for a given hostname, an error will be generated.

If the *prefixlen* is not specified, the *netmask* is computed using the order specified for the *netmasks* line */etc/nsswitch.conf*. If that fails an error will be generated asking the user to specify prefixlen explicitly.

Note: If the interface requires only a local address, specify it directly with the -a option as follows: '-a <addr>[/prefixlen]'. The address will automatically be considered a local address.

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 local addr, but if any *netmask* is provided an error will be generated..

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 5.1.2).

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 5.1.2. Note that the default link-local may be generated without turning on IPv6 ADDRCONF by following the steps show in Section 5.1.2

The address is marked *up* 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 be marked *down* and hence cannot be used 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 instantiated 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.

Examples:

Create a point-to-point interface with local address as 12.1.2.3 and remote address as 12.1.2.4

 # ipadm create-addr -T static -a local=12.1.2.3,remote=12.1.2.4 tun0/mytunaddr

Create a static IPv4 address (11.22.33.44/24) on net0 IP interface

 # ipadm create-addr -T static -a local=11.22.33.44/24 net0/netaddr

5.1.2 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* subcommand.

 # ipadm create-addr [-t] -T addrconf [-i <interface_id>] [-p {stateless | stateful} = {yes | no},...] <addrobj>

This subcommand creates auto-configured 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 as per RFC 4291. 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 auto-configured based on prefixes advertised by routers as described in RFC 4862. When *stateless=no* is specified, stateless auto-configuration based on advertised prefixes will not be performed, however the auto-configuration as in RFC 4861 will still take place (to configure the default routers and the on-link prefix list).

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.

All four possible combination of *stateful* and *stateless* is valid and by default both are *yes*.

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

The default IAID (Identity Association Identifier) for DHCPv6 is the *ifIndex* number of ifname. Future extensions to the *create-addr -T addrconf* subcommand will allow a custom IAID to be provided via the -I option.

The default DUID (DHCP Unique Identifier) for DHCPv6 is LLT (Link Layer address plus time) if possible, otherwise we use the *libuuid(3LIB)* 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, a message is sent to *in.ndpd t*o start the auto-configuration and it 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 addresses for the received prefixes using the Interface ID that was sent by *ipadm*, depending on whether stateless and stateful auto-configuration were enabled.

When *ndpd.conf(4)* settings have been customized manually, and are in conflict with those selected by ipadm(1m), the values provided with ipadm will override the *ndpd.conf* settings. In particular 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 *ipadm show-addr* command can be used to view the addresses associated with an *addrobj* of type *addrconf*.

NOTE: As part of creating *addrconf* address objects, we have to create link-local address on the given interface using the default interface-id or the user-provided interface-id (through -i). If we use SIOCSLIFADDR to generate this link-local address, the *ill->ill_manual_linklocal* will be set. With that

bit set any later changes to interface-id (through *ifconfig <ifname> inet6 token <token>)* wouldn't be reflected in the link-local address and that is not the expected behavior. Therefore, a new Consolidation Private ioctl, SIOCSLIFPREFIX, had to be introduced, which generates IPv6 addresses using the given prefix and the ill->ill_token and this does not set the *ill->ill_manual_linklocal* bit. This should now allow any changes to interface-id to be reflected in link-local address.

Examples:

The example below tells *in.ndpd* to auto-configure stateless and stateful IPv6 addresses and that *::abcd/64* should be used as the token for those addresses.

 # ipadm create-addr -T addrconf -i ::abcd/64 net0/v6addrs

The example below would generate the default link-local on net0 without enabling stateless or stateful address configuration:

 # ipadm create-addr -T addrconf -p stateless=no,stateful=no net0/llonly

5.1.3 IPv4 DHCP address management

 # ipadm create-addr [-t] -T dhcp [-w <seconds> | forever] <addrobj>

This subcommand 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. By default this operation is persisted, unless -t option is specified.

The -w option is used to specify the maximum amount of time to wait before the command completes. By default, the wait time is 120 seconds.

Examples:

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

5.2 Displaying IP addresses on the system

 # ipadm show-addr [[-p] -o <field>,...] [<addrobj>]

This subcommand will display all the IP addresses available on the system (including the ones in the persistent store) or if the *addrobj* is specified, it just displays the address identified by *addrobj*. For every address object, following fields will be displayed:

Note: By default ADDROBJ, TYPE, STATE and ADDR will be displayed. CURRENT and PERSISTENT will be displayed when -o all is specified in i*padm show-addr*

- ADDROBJ: The name of the address object.
- TYPE: The type of the address 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: Indicates the status of the address and can be any of the values captured in Table 3. The states are listed in the order they are determined.
- CURRENT: For address objects in active configuration, it can be any of the values captured in Table 4
- PERSISTENT: Specifies the values that will be applied when the address object is instantiated

on reboot or re-enabled using subcommands in Section 6. It can be any or all of U, p & d

- ADDR: Numeric IPv4 or IPv6 address. In the case of point-to-point interfaces, the addresses of both the endpoints, are printed (laddr --> raddr).

State	Semantics
disabled	Indicates that the address has been disabled from the active configuration using 'disable-addr' subcommand.
duplicate	Indicates that the address is a duplicate.
down	Indicates that the address is marked down.
tentative	Indicates the Duplicate Address Detection is in progress.
ok	Indicates that the address is up.
inaccessible	Indicates that the address is not accessible because the interface on which this address is created is failed (IFF_RUNNING is cleared)

Table 3: Address STATE information as returned by ipadm show-addr

Flags	Flag name	Semantics
U	up	Address is marked *up* (i.e., IFF_UP is set)
u	unnumbered	address matches the local address of some other link in the system
p	private	address not advertised by the routing daemon
t	temporary	temporary IPv6 address as defined in RFC 3041
d	deprecated	Address should no longer used as a source address in new communications, but packets addressed to such an address are processed as expected.

Table 4: Address FLAG information as returned by ipadm show-addr

To determine that DAD has been initiated, we have introduced a new socket ioctl *SIOCGLIFDADSTATE*, which retrieves the *ipif_t*'s *ipif_addr_ready* bit. If it is set to 0, then DAD is in progress. This is a transient state until either address becomes *duplicate* or *up*.

Note: The STATE names, *inaccessible, tentative* and *duplicate* were influenced by RFC 4293 (Management Information Base for the Internet Protocol (IP)).

Examples:

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

Create *tunaddr2* on *tun0*, with local endpoint being *10.8.48.242* and remote endpoint being *10.8.48.1*

```
# ipadm create-addr -T static -a local=10.8.48.242,remote=10.8.48.1 tun0/tunaddr2
```

Create an *addrobj* named *bge0/addr1*, in *down* state, with *static* address *10.8.48.170* and *prefixlen* 25. Bring *UP* the address using *up-addr* temporarily and *deprecate* the address using *set-addrprop*.

```
# ipadm create-addr -T static -d -a local=10.8.48.170/25 bge0/addr1
# ipadm up-addr -t bge0/addr1
# ipadm set-addrprop -p deprecated=yes bge0/addr1
```

```
# ipadm show-addr -o all
ADDROBJ           TYPE     STATE    CURRENT   PERSISTENT  ADDR
lo0/v4            static   ok       U-----    ---         127.0.0.1/8
vni0/vniaddr      static   disabled -----     U--         11.2.3.1/8
bge0/dhcpaddr     dhcp     ok       U-----    –           10.8.48.173/25
bge0/addr1        static   ok       U-----    --d         10.8.48.170/25
tun0/tunaddr2     static   ok       U-----    U--         10.8.48.242-->10.8.48.1
lo0/v6            static   ok       U-----    ---         ::1/128
bge1/auto         addrconf ok       U-----    ---         fe80::203:baff:fe44:3d35/10
bge1/auto         addrconf ok       U-----    ---         2002:a08:39f0:3:203:baff:fe44:3d35/64
bge2/_d           static    ok      U-----     ---          172.16.11.2/24
```

The *bge0/addr1* is marked *up* in running configuration however it is not *up* in saved configuration. So on reboot, it will be instantiated as *down* and *deprecated* flag set. The *vni0/vniaddr* is present only in saved configuration and when it is instantiated on reboot or re-enabled using the subcommands in Section 6, it will be marked *up*. Further, some of the address objects, that were created outside of *ipadm(1M)*, for example, using *ifconfig(1M)*, have names starting with '_' and they are generated by the library automatically and can be used in various address related *ipadm* subcommands.

5.3 Deleting an address on an interface

 # ipadm delete-addr [-r] <addrobj>

This subcommand removes the address identified by *addrobj* on the implied interface. Note that name-resolution routines are not consulted for deleting the address. This command is persistent, i.e., an address deleted by *ipadm delete-addr* updates the persistent store. See Section 6 for how to disable an address in the running configuration.

If a*ddrobj* is of type dhcp, then the dhcp lease will be dropped by default on using *delete-addr*. If the option -r is used, then the lease will be released. This option does not apply to address objects of type *static* and *addrconf*.

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 5.3.1

5.3.1 Subtleties in address deletion

As mentioned earlier, the implementation choices made in the Solaris kernel impose some constraints 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 administratively down. 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.

A detailed discussion of the problem and some long-term solutions are available here[4]. As described in that proposal, *ipadm* and *libipadm.so* will emulate deletion of the first address by replacing it with INADDR ANY or ::.

5.4 getifaddrs(3SOCKET)

The acquisition of 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, Solaris only provides the harder-to-use SIOCGLIFCONF ioctl. Therefore,

- the *getifaddrs()* library API will be ported to Solaris and made available through *libsocket*. The input/output semantics will be the same as that described FreeBSD man page[5], except that the interface name will be a *logical interface name* (e.g., *net0:5*) that can be used in other socket ioctl calls. The reported address information may contain *AF_LINK* addresses in the future, so the caller will not make assumptions about the address type. Only addresses that are up will be reported.
- *ipadm_addr_info()* will return IP address information for all IP addresses on the system, including those which are ~IFF UP. However INADDR ANY or :: holes left from the deletion of the first address, will be excluded. Additionally, *ipadm_addr_info()* will also report TYPE and STATE as described in Section 5.2. The interface name returned by the library call will be the logical interface name.

5.5 Refreshing IP Address

 # ipadm refresh-addr [-i] <addrobj>

This subcommand 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 auto-configured address object then we return error.

Also, when -i is provided, the subcommand will obtain network configuration parameters from DHCP, for that IP address without obtaining a lease on it (equivalent to *ifconfig dhcp inform)*

5.6 Marking IP Address down

 # ipadm down-addr [-t] <addrobj>

This subcommand is provided to mark the IP address identified by *addrobj, down,* so that the the system does not attempt to use the address assigned to that interface as a source address for outbound packets and will not recognize inbound packets destined to that address as being addressed to this host. It's an inexpensive way to mark the address down without removing it from the kernel altogether. Down addresses can be useful in cluster configurations where a standby node should be able to quickly take over from the master when the master fails.

This subcommand has no effect if the *addrobj* was already *down* prior to the *down-addr* invocation. Since IPv6 auto-configured addresses are managed by other protocols, they may not be marked *down* by this command, and an error is generated if *addrobj* is of type '*addrconf* '. This subcommand persists the state, by default. This behavior can be overridden using -t.

4 "GLIFCONF/IFF_UP" . http://mail.opensolaris.org/pipermail/brussels-dev/2009-May/001527.html.
5 "FreeBSD Man Pages: GETIFADDRS(3)" http://www.freebsd.org/cgi/man.cgi?query=getifaddrs.

5.7 Marking IP Address up

 # ipadm up-addr [-t] <addrobj>

This subcommand is provided to mark the IP address identified by *addrobj, up*. By default all addresses are created, marked *up*. If the *addrobj* is of the type *static or dhcp* then the address identified by the address object is marked *up*, so that it can be used as a source/destination of outbound/inbound packets. This command has no effect if the *addrobj* has been marked *down* by the system because it is a duplicate address, or if the address was *up* prior to the *up-addr* invocation. Since IPv6 auto-configured addresses are managed by other protocols, they may not be marked *up* by this command, and an error is generated if *addrobj* is of type '*addrconf* '. This subcommand persists the state, by default. This behavior can be overridden using -t.

5.8 Address Properties

 # ipadm set-addrprop [-t] -p <prop>=<value[,...]> <addrobj>
 # ipadm reset-addrprop [-t] -p <prop> <addrobj>

The *set-addrprop* subcommand is used to modify an address property to the value specified by the user. If the property takes multiple values then the values should be specified with comma as the delimiter. If the operation is successful, the value will be persisted, unless -t option is specified. The persisted value is reapplied on reboot.

The *reset-addrprop* subcommand is used to reset an address property to it's default value. If the operation is successful, the persisted value will be purged from the datastore, unless -t option is specified.

For both of the above subcommands, only one property can be modified at a time. Further if the address object is of type *addrconf* modifying address properties is not allowed since these addresses are managed by other protocols and daemons (*in.ndpd).*

Following are the list of address properties that will be supported: *deprecated, prefixlen, private, transmit, broadcast* and *zone*. For definition of these properties, please consult *ipadm* man page.

 # ipadm show-addrprop [[-c] -o <field>[,...]] [-p <prop>,...] [<addrobj>]

This subcommand retrieves the value of the given address property from the kernel and from the persistent store. Several properties of interest can be retrieved at a time by providing comma separated property names to -p option.

Providing an *addrobj* is optional. When not provided, for all the addresses configured on the system **using *ipadm***, the address properties will be displayed.

For each address property, following fields are displayed in the output:

- ADDROBJ - Name of the address object
- PROPERTY - Name of the property
- PERM - Specifies if the value is read only, write only or read and write
- CURRENT - The current value of the property
- PERSISTENT- The persistent value of the property
- DEFAULT - Default value of the property
- POSSIBLE - Possible range of values for a given property

Examples

```
# ipadm create-addr -T static -a local=10.10.1.2/24 net0/addr1
# ipadm show-addrprop net0/addr1
ADDROBJ       PROPERTY   PERM  CURRENT  PERSISTENT  DEFAULT     POSSIBLE
net0/addr1    deprecated rw    off      off         off         on, off
net0/addr1    prefixlen  rw    24       24          8           0-32
net0/addr1    private    rw    off      off         off         on, off
net0/addr1    transmit   rw    on       on          off         on, off
net0/addr1    zone       rw    global   --          global      –
```

Note: To modify the zone assignment such that it persists across reboots, *zonecfg(1M)* must be used.

```
# ipadm set-addrprop -t -p deprecated=on net0/addr1
# ipadm show-addrprop -p deprecated net0/addr1
ADDROBJ       PROPERTY   PERM  CURRENT  PERSISTENT  DEFAULT     POSSIBLE
net0/addr1    deprecated rw    on       off         off         on, off
# ipadm reset-addrprop -p deprecated net0/addr1
# ipadm show-addrprop -p deprecated net0/addr1
ADDROBJ       PROPERTY   PERM  CURRENT  PERSISTENT  DEFAULT     POSSIBLE
net0/addr1    deprecated rw    off      --          off         on, off
```

Since the PERSISTENT is '--' it means that the value was purged in the data-store. If an object property was reset it is not required to persist the default value because the objects are created with default values, when there are no persistent values for those object properties.

6 Disabling/Enabling of objects from running/saved configuration

When an *object* is instantiated for the first time, there is no prior state for the *object* in the saved configuration managed by the *ipadm* subsystem, and the *object* can be created afresh. However, after an *object* is created and configured, if the instance of the *object* is removed from the running-configuration (e.g., via '*ifconfig <object> unplumb*'), we end up in a state where the *object* is missing in the running configuration, but there is a stored state for the *object* in the saved configuration. We refer to such objects as *disabled* object.

Although both the *dladm* and *ipadm* sub-systems are capable of encountering a state where the *object* exists, but is hidden in the running configuration, *ipadm* is more susceptible to this state due to the interactions with *ifconfig* changes that apply only to the running configuration. An attempt to create the same *object* via *ipadm* when the system is in this state would (correctly) fail, since the *object* exists in the saved configuration. In order to restore the running configuration for the *object* at this point, we would need to *enable* the *object*.

If an interface or an address was temporarily deleted using *'ifconfig unplumb <ifname>[:lifnum]'* or *'ipadm disable-if or ipadm disable-addr'*, then to restore this interface or address, one would use *'ipadm enable-if' or 'ipadm enable-addr'*. In *dladm(1M)* if an object (*vnic, aggregation, et al)* was temporarily deleted then there is no way to restore them, unless you reboot the machine. In *ipadm(1M)*, we have *'enable-*'* subcommands to restore persistent configuration and it's used on reboot to re-instantiate the persistent settings.

In this section we will look at those commands which can be used to *disable* an *object* from the running configuration and later on *enable* the same *object* from the saved configuration. Further -t is mandatory for all of these subcommands and means that *enable/disable* is a temporary operation.

 # ipadm disable-addr -t <addrobj>

This subcommand disables the address object by removing the address from the running configuration. It is equivalent to ***'ifconfig <lifname> unplumb'.*** Further the persistent configuration is untouched.

 # ipadm disable-if -t <interface>

This subcommand disables the interface object by removing the given interface and all the addresses configured on that interface, from the running configuration. The persistent configuration is untouched. This subcommand is equivalent to '***ifconfig <interface> unplumb'.***

 # ipadm enable-if -t <interface>

This subcommand enables the given interface by reading the configuration from the persistent store. All the persistent interface properties, if any, are applied and all the persistent addresses, if any, on the given interface will be enabled.

 # ipadm enable-addr -t <addrobj>

This subcommand enables the given address object by reading the configuration from the persistent store. All the persistent address properties are applied to the address object. This subcommand requires that the interface on which the address object is being created be present. If the interface itself is missing in running configuration and is present in persistent store, then the user has to run *ipadm enable-if* first.

Examples:

```
# ipadm create-addr -T static -a local=10.2.3.4/24 bge1/v4static
# ipadm set-addrprop -p private=yes bge1/v4static
# ipadm show-addr -o all bge1/v4static
ADDROBJ       TYPE   STATE CURRENT   PERSISTENT          ADDR
bge1/v4static static ok    U-p--      Up-                 10.2.3.4/24
```

Disable the address object bge1/v4static

```
# ipadm disable-addr -t bge1/v4static
# ipadm show-addr bge1/v4static
ADDROBJ       TYPE   STATE     ADDR
bge1/v4static static disabled  10.2.3.4/24
```

Disable the interface object bge1

```
# ipadm disable-if -t bge1
# ipadm show-if bge1
IFNAME   STATE      CURRENT PERSISTENT
bge1     disabled   ----------      -46
```

Enable the interface object from the persistent configuration

```
# ipadm enable-if -t bge1
# ipadm show-if bge1
IFNAME   STATE  CURRENT PERSISTENT
bge1     ok     bm-------46  -46
#ipadm show-addr bge1/v4static
ADDROBJ       TYPE   STATE   ADDR
bge1/v4static static ok      10.2.3.4/24
```

Note that when the interface object was enabled all the address objects configured on that interface will be enabled too.

7 Global Protocol Property management

This section describes a set of *ipadm* subcommands which can be used to modify protocol properties temporarily or persistently and view the active or persistent property values. These subcommands provide TCP/IP tunable management equivalent to the features of the */sbin/ndd* command to be applied on the appropriate protocol (currently tcp, ip, udp and sctp)

Note that for some IP tunables, it may be possible to set the value of the property both globally, and on a per-interface basis. The per-interface value may be set using the *set-ifprop* subcommand described in Section 4.4. In such cases, if the administrator chooses to customize the per-interface value of the tunable to be distinct from the global value, the per-interface value overrides the global setting for that interface. However, if the global value is changed after changing the per-interface value then the per-interface will be overwritten. Therefore, whenever the per-interface value needs to be distinct from global value, it must always be changed after changing the global value.

7.1 Modifying protocol properties

 # ipadm set-prop [-t] -p <prop>[+|-]=<value[,...]> <protocol>
 # ipadm reset-prop [-t] -p <prop> <protocol>

The *set-prop* subcommand is used to modify a protocol property to the value specified by the user. If the property takes multiple values then the values should be specified with comma as the delimiter. If the operation is successful, the value will be persisted, unless -t option is specified. The persisted value is reapplied on reboot. Only one property can be modified at a time.

The *reset-prop* subcommand is used to reset a *protocol* property to it's default value. If the operation is successful the persisted value will be purged from the datastore, unless -t option is specified. Only one property can be reset at a time.

The *set-prop* subcommand also provides 'qualifiers' to perform add/delete 'aka' +/- in addition to assignment.

- + : adds the given value to the current list of value(s)
- - : removes the given value from the current list of value(s)
- = : makes a new assignment and removes all the current value(s).

For an example, see *extra_priv_ports* entry in Table 5.

When we last checked, in *snv_129*, there were 82 IP tunables, 67 TCP tunables, 16 UDP tunables and 42 SCTP tunables. Obviously not all the tunables will be C*ommitted* as part of the first phase. Very few tunables, which are deemed important, will be C*ommitted*. However those tunables which do not get C*ommitted* will still be supported as a **private** property. These private properties will retain the same name as they have in *ndd*. For e.g., *tcp_wscale_always*, *tcp_tstamp_always*, *arp_probe_count*, *ip_defend_interval*, et al. Like other C*ommitted* properties the private properties can be modified, reset and retrieved. They can also be persisted so that it will be reapplied on reboot. However *private* properties will not appear in *'ipadm show-prop <protocol>'* output along with C*ommitted* properties and they are subject to change in future release.

Following global protocol properties will be C*ommitted.*

Property name	Protocol	Meaning
ttl	IPv4	Specifies the value that will be set for *ttl*/*hoplimit* field of IPv4 or IPv6 header. Can be used to prevent the system from reaching other systems more than N hops away where N was the value specified.
hoplimit	IPv6	
forwarding	IPv4	Enable/disable global IPv4 or IPv6 forwarding. All the configured interfaces will start/stop forwarding packets. Individual interfaces can override the global option using *set-ifprop*. See Section 4.4
	IPv6	
sack	TCP	Selective acknowledgment (SACK) allows recipients to selectively acknowledge out-of-sequence data and is intended to increase performance for data transfers over lossy links. See RFC 2018 for information on the SACK. Possible values and meaning: • *never* – will not accept SACK or send out SACK information • *passive* – will accept SACK but not send out • *active* – will both accept SACK and send out SACK information
ecn	TCP	Explicit Congestion Control (See RFC 3168 for more information). Possible values are same as above; *never*, *passive* and *active*
smallest_anon_port largest_anon_port	TCP SCTP UDP	These options define the upper and lower bounds on ephemeral ports. Ephemeral (means short-lived) ports are used when establishing outbound network connections.
smallest_nonpriv_port	TCP SCTP UDP	This option define the start of non-privileged ports. The non-privileged port range normally starts at 1024. Any program that attempts to bind a non-privileged port does not have to run as root.
extra_priv_ports	TCP SCTP UDP	This option define additional privileged ports outside of the 1-1023 range. Any program that attempts to bind the ports listed here must run as root. This prevents normal users from starting server processes on specific ports. These ports can be added/removed/assigned using the modifiers +/-/= Example: **Add 1047, 1048, 1049 and 1050 as extra privileged ports for tcp.** # ipadm set-prop -p extra_priv_ports=1047 tcp # ipadm set-prop -p extra_priv_ports+=1048 tcp # ipadm set-prop -p extra_priv_ports+=1049 tcp # ipadm set-prop -p extra_priv_ports+=1050 tcp Delete 1048 as extra privileged port. # ipadm set-prop -p extra_priv_ports-=1048 Display all the extra privileged ports for tcp. # ipadm show-prop -p extra_priv_ports tcp PROTO PROPERTY PERM CURRENT PERSISTENT DEFAULT POSSIBLE ipv4 extra_priv_ports rw 1047,1049, 1050 1047,1049, 1050 2049,4045 1024-65535
recv_maxbuf send_maxbuf	ICMP, TCP, SCTP, UDP	This property modifies the receive or send buffer sizes for the given protocol. (recv_maxbuf aka SO_RECVBUF, affects system wide, while send_maxbuf aka SO_SNDBUF, affects system wide).

Table 5: Committed Protocol properties

 # ipadm show-prop [[-c] -o <field>[,...]] [-p <prop>[,...]] [<protocol>]

This subcommand retrieves the value of the given protocol property from the kernel and from the persistent store. Several properties of interest can be retrieved at a time by providing comma separated property names to -p option.

Providing a protocol name is optional. When not provided, for all the protocols supported by libipadm.so, the specified or all committed properties will be displayed. For each property, following fields are displayed in the output.

- PROTO - Specifies the protocol to which the property belongs
- PROPERTY - Name of the property
- PERM - Specifies if the value is read only, write only or read and write
- CURRENT - The current value of the property
- PERSISTENT - The persistent value of the property
- DEFAULT - Default value of the property
- POSSIBLE - Possible range of values for a given property

Examples:

Change the tcp's smallest non privilege port to 2048 from it's default value of 1024

#ipadm set-prop -p smallest_nonpriv_port=2048 tcp

Turn on IPv4 forwarding temporarily on all the IP interfaces

#ipadm set-prop -t -p forwarding=on ipv4

Display the above properties from running config

 # ipadm show-prop -p smallest_nonpriv_port tcp

PROTO	PROPERTY	PERM	CURRENT	PERSISTENT	DEFAULT	POSSIBLE
tcp	smallest_nonpriv_port	rw	2048	2048	1024	1024-32768

 # ipadm show-prop -p forwarding ipv4

PROTO	PROPERTY	PERM	CURRENT	PERSISTENT	DEFAULT	POSSIBLE
ipv4	forwarding	rw	on	off	off	on, off

7.2 ndd(1M) and libipadm.so interactions

The *ndd(1M)* utility has been changed to use the Consolidation Private interfaces exposed by *libipadm.so* library. The *ndd(1M)* will continue to work as it used to before and will support all the properties it used to support, without persistence. The user shouldn't see any difference whatsoever. However, if the user uses new *ipadm* property name (*forwarding*, *smallest_anon_port*, et al) an error will be thrown, asking the user to use *ipadm.*

7.3 System call details for property management

ndd(1M) uses Mentat's *ND_SET* and *ND_GET* ioctls 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 ioctl interface to set and get properties. Two socket ioctl's, SIOCSETPROP and SIOCGETPROP are defined to set/get property values respectively. Following structure will be used as an argument to the aforementioned ioctls.

```
typedef struct mod_ioc_prop_s {
    uint_t          mpr_version;
    uint_t          mpr_flags;    /* primarily used in GET. Determines the type of value to get */
    char            mpr_ifname[LIFNAMSIZ];      /* interface name if we are interested in  it's prop */
    uint_t          mpr_proto;          /* protocol to which the property belongs */
    char            mpr_name[MAXPROPNAMELEN];
    uint_t          mpr_valsize;
    char            mpr_val[1];  /* input parameter for SET, output parameter for GET */
} 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. The project team has done exhaustive search and came to a conclusion that ND_SET/ND_GET ioctls are being used for the purpose of setting/getting ip_forwarding and ip6_forwarding ndd variables. So we will have to add special code to support ND_SET/ND_GET of just these two tunables.

8 libipadm.so library API details

Note: None of the API's discussed here are stable and are subject to change. They are Consolidation Private interfaces and they are discussed here for the completeness of the design document.

Every *ipadm* subcommand calls into a function in *libipadm.so* library. There is a 1-1 mapping between an ipadm subcommand invocation and a library interface. We will look at the relevant *libipadm.so* functions in this section. All the API's exported by the library are reentrant and hence multithread safe.

8.1 Opaque Structures

The consumers of *libipadm.so* must first open a handle *ipadm_handle_t* into the library by calling *ipadm_open()*. Once all the desired invocations are completed the handle must be closed using *ipadm_close()*. The *ipadm_handle_t* is an opaque structure and contains socket file descriptors, door descriptor to *ipmgmtd* daemon, handle to libdladm and private flags.

```
struct ipadm_handle {
    int                 iph_sock;       /* socket descriptor for v4 interface */
    int                 iph_sock6;      /* socket descriptor for v6 interface */
    int                 iph_door;       /* door descriptor to ipmgmtd */
    dladm_handle_t      iph_dlh;        /* handle to libdladm library */
    uint32_t            iph_flags;      /* internal flags */
    pthread_mutex_t     iph_lock;       /* lock to set door_fd */
};
```

All of the API's that implement *ipadm* subcommand take *ipadm_handle_t* as the first argument and they return *ipadm_status_t* (they are error codes returned by library and if it's not *IPADM_SUCCESS*, then it means there was an error while executing the API. *ipadm_status2str(status)* can be used to convert the error status to a meaningful error message).

Another opaque data structure that is used inside the library is *ipadm_addr_t*. This data structure represents *addrobj* and holds ifname, aobjname, address type (static, dhcp or addrconf), logical interface number on which the address is configured, a union which can contain dhcp, static or addrconf related information based on the address type. This structure needs to be passed for address operations such as creation, deletion, up/down and optionally for show-addr.

```
struct ipadm_addr_s {
    char                    ipadm_ifname[LIFNAMSIZ];
    int32_t                 ipadm_lifnum;
    char                    ipadm_aobjname[IPADM_ADDROBJ_LEN];
    ipadm_addr_type_t       ipadm_addr_type;
            union {
        struct u_ipadm_static_addr {
            char                        ipadm_ahname[MAXNAMELEN];
            struct sockaddr_storage     ipadm_addr;
            uint32_t                    ipadm_masklen;
            char                        ipadm_dhname[MAXNAMELEN];
            struct sockaddr_storage     ipadm_dstaddr;
        } u_ipadm_static_addr;
        struct u_ipadm_ipv6_intfid {
            struct sockaddr_in6         ipadm_intfid;
            uint32_t                    ipadm_intfidlen;
            boolean_t                   ipadm_stateless;
            boolean_t                   ipadm_stateful;
        } u_ipadm_ipv6_intfid;
        struct u_ipadm_dhcp_s {
            boolean_t                   ipadm_primary;
            int32_t                     ipadm_wait;
        } u_ipadm_dhcp;
    } ipadm_addr_u;
};
```

Since this is an opaque structure, *libipadm.so* exports interfaces to create and delete this structure, *ipadm_create_ addrobj(ipadm_addr_type_t)* and *ipadm_destroy_addrobj(ipadm_addr_t)*. There are few other API's used to set/get some of the relevant fields of *ipadm_addr_t*. An example should make things clear:

A pseudo-code to create a static address, *10.10.11.1/24*, marked *up* on *net0* with '*net0/addr1*' as the *aobjname*, programmatically, would be:

```
ipaddr = ipadm_create_addrobj(type = IPADM_ADDR_STATIC, "net0/addr1")
ipadm_set_addr(ipaddr, "10.10.11.1/24", AF_UNSPEC) or
ipadm_set_addr(ipaddr, "zhadum.east.sun.com", AF_INET)
ipadm_create_addr(handle, ipaddr,  up = B_TRUE,  IPADM_OPT_ACTIVE)
```

In the first phase of this project, *ifconfig(1M)* has been modified to use *libipadm.so* API's to create STATIC addresses only. With more support in the library being added in the subsequent integrations, library API's will be used in *ifconfig(1M)* for DHCP and IPMP. In *ipadm_create_addrobj()*, minimally *ifname* has to be provided. If the optional address object name is not provided then the library will generate an address object name automatically and such names will begin with '_'. These names start to be single letters *(_a, _b, _c, et al, as in bge1/_a, bge1/_b and bge1/_c)* and monotonically increase and after _z, they wrap into *_aa, _ab, _ac, et al.*

8.2 Important Data structures and API's

The set of IP interfaces configured on the system can be retrieved using *ipadm_show_if()* (See Table 6). It returns *ipadm_if_info_t*, which is defined below.

```
typedef struct ipadm_if_info_s {
    struct ipadm_if_info_s      *ifi_next;
    char                        ifi_name[LIFNAMSIZ]; /* name of the interface */
    uint64_t                    ifi_flags;           /* IFF_* flags */
} ipadm_if_info_t;
```

The set of addresses configured on the system can be retrieved using *ipadm_addr_info()* (See Table 6). It returns *ipadm_addr_info_t*, which is defined below.

```
typedef struct ipadm_addr_info_s {
    struct ifaddrs              ia_ifa;                         /* see section 5.4 */
    char                        ia_sname[NI_MAXHOST];           /* hostname of local end-point */
    char                        ia_dname[NI_MAXHOST];           /* hostname of remote end-point */
    char                        ia_aobjname[IPADM_ADDROBJ_LEN]; /* address object name */
    ipadm_addr_type_t           ia_type;                        /* type of address (static, dhcp or addrconf )*/
    ipadm_addr_state_t          ia_stat                         /* see Table 3 */
    uint_t                      ia_flags;                       /* see Table 4 */
} ipadm_addr_info_t;
```

ipadm subcommand	Library API invoked
Interface Management	
create-if	*ipadm_create_if(handle, ifname, flags)*
delete-if	*ipadm_delete_if(handle, ifname, flags)*
show-if	*ipadm_if_info(handle, ifname, ipadm_if_info_t **, flags, lifc_flags)*
set-ifprop	*ipadm_set_ifprop(handle, ifname, pname, pval, proto, flags)*
reset-ifprop	*ipadm_set_ifprop(handle, ifname, pname, pval, proto, flags)* *flags = IPADM_OPT_DEFAULT*
show-ifprop	*ipadm_get_ifprop(handle, ifname, pname, buf, bufsize, proto, flags)*
Address management	
create-addr -T static	*ipadm_create_addr(handle, ipadm_addr_t, boolean_t up, flags)* *ipadm_addr_t - IPADM_ADDR_STATIC*
create-addr -T dhcp	*ipadm_create_addr(handle, ipadm_addr_t, boolean_t up, flags)* *ipadm_addr_t - IPADM_ADDR_DHCP*
create-addr -T addrconf	*ipadm_create_addr(handle, ipadm_addr_t, boolean_t up, flags)* *ipadm_addr_t - IPADM_ADDR_IPV6_ADDRCONF*
delete-addr	*ipadm_delete_addr(handle, ipadm_addr_t)*
show-addr	*ipadm_addr_info(handle, aobjname, af, ipadm_addr_info_t **, flags, lifc_flags)*
down-addr	*ipadm_down_addr(handle, ipadm_addr_t)*
up-addr	*ipadm_up_addr(handle, ipadm_addr_t)*
refresh-addr	*ipadm_refresh_addr(handle, ipadm_addr_t)*
set-addrprop	*ipadm_set_addrprop(handle, pname, pval, aobjname, flags)*
reset-addrprop	*ipadm_set_addrprop(handle, pname, pval, aobjname, flags)* *flags = IPADM_OPT_DEFAULT*
show-addrprop	*ipadm_get_addrprop(handle, pname, buf, bufsize, aobjname, flags)*
Deleting objects from running configuration and restoring them from saved configuration	
disable-addr	*ipadm_disable_addr(handle, ipadm_addr_t, flags)*
disable-if	*ipadm_disable_if(handle, ifname, flags)*
enable-if	*ipadm_enable_if(handle, ifname, flags)*
enable-addr	*ipadm_enable_addr(handle, ipadm_addr_t, flags)*
Global protocol property management	
set-prop	*ipadm_set_prop(handle, pname, pval, proto, flags)*
reset-prop	*ipadm_set_prop(handle, pname, pval, proto, flags)* *flags |= IPADM_OPT_DEFAULT*
show-prop	*ipadm_get_prop(handle, pname, buf, bufsize, proto, flags)*

Table 6: ipadm subcommands to libipadm API mapping

9 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*. Specific details of the structure of the sub-commands will be provided when the component is delivered.[6] In the first push, we do not allow creating addresses or setting interface properties on either IPMP meta-interface or underlying interface .

10 ipmgmtd daemon and IP interface management SMF service

10.1 ipmgmtd

This project adds a new daemon *ipmgmtd* which keeps track of internal state of the library *libipadm.so* and also provides a means to update the *ipadm* persistent/temporary data-store. The communication between the library *libipadm.so* and the *ipmgmtd* is through *door_call(3c)*. The *ipmgmtd* daemon performs the tasks listed below.

- For every *addrobj* created by *libipadm* the daemon adds the *addrobj* into a *list addrobjmap* and stores the following information for that *addrobj*:
  - logical interface number: *lifnum* on which the address is created.
  - address type: static, dhcp or addrconf
  - address family: inet or inet6
  - flags: if *addrobj* is in active and/or persistent store

  This information is used by the library in subsequent operations on the *addrobj* (mapping *addrobj* to *lifname*, knowing lifname is essential for all interface operations, et al,.)

- 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 daemon updates persistent data-store with IP interface configuration and TCP/IP property values, so that these values can be reapplied when the system 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 10.2. Since the daemon is a door server, it could possibly serve multiple requests at a time. So access to the persistent store is achieved using a *pthread_rwlock_t* and access to the *list addrobjmap* is synchronized using a *pthread_rwlock_t*. The daemon as such has one thread, which is the *main* thread.

10.2 IP interface management SMF service

A new SMF service *svc:/network/ip-interface-management:default* has been created to start the *ipmgmtd* daemon. The daemon **must** to be started before any other networking service which configures IP interfaces is started, so as to ensure that the persistent IP interface configuration are reapplied correctly and any changes to the active configuration are tracked correctly. The mechanism for restoring persistent properties is described in Section 11.

6 Initial proposal for these subcommands are available here: http://arc.opensolaris.org/caselog/PSARC/2009/306/inception.materials/brussels2_design.pdf.

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:default*. The *ip-interface-management* service is not dependent on any other SMF service.

10.3 Exclusive-IP zone support

In each exclusive-IP zone, the *svc:/network/ip-interface-management:default* 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.

11 Mechanism for restoring Persistent information

There are two aspects to restoring stored configuration information:

1. re-application of global protocol properties (that applies globally to *TCP*/*UDP*/*IP*/*SCTP* modules), described in Section 11.1
2. re-application of interface configuration, including IP addresses, per-interface properties etc., described in Section 11.2

11.1 Persistence of properties global to all interfaces

Initial proposal was to use */sbin/netstart* to restore persistent protocol tunables during boot. This process would be started by *init(1M)*, by reading */etc/inittab*, before *svc.startd(1M)* comes up. The idea was to restore settings close to the execution of `*soconfig(1M)*' (*soconfig* maps sockets to service providers and networking applications are useless without the execution of *soconfig)*

However, modifying */etc/inittab* in the post-SMF world was something that was not well received and also with IPS obsoleting SVR4 post-install scripts, modifying */etc/inittab* during the upgrade path was not possible. Therefore, we will now restore the persistent protocol tunables from a SMF script that starts the `ipmgmtd' daemon. Further `ipmgmtd' daemon is one of the first networking services to come up and it comes up even before *network/loopback*. That way the protocol properties would be re-instantiated before any of the IP interfaces are plumbed and before any of the networking applications starts.

A project private interface, *ipadm init-prop*, will be used to re-instantiate protocol properties at boot time. It will be called from *net-ipmgmt* SMF script that also starts *ipmgmt* daemon.

11.2 Restoration of interface configuration

A rudimentary mechanism for providing persistent configuration of interfaces exists through the files in */etc* such as */etc/hostname.interface*. 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* 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 persistent *ipadm* configuration information

12 Security Considerations

12.1 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 IP interface configuration.

12.2 auth_attr(4)

None of the authorizations defined in */etc/security/auth_attr* satisfy our requirement. They are mostly related to nwam, reading or writing */etc/hosts*, link security and wifi configuration. We will add *solaris.network.interface.config* authorization - which is required to configure network interfaces. Verification of this authorization will be done inside the library, *libipadm.so*. This authorization will be added to *Network Management* profile.

Further for viewing interface, address or property configurations we don't need this authorization, as ordinary users, today, can already view such configuration.

12.3 exec_attr(4)

ipadm would need *sys_ip_config* privilege to configure system's IP interfaces, addresses and protocol properties. So we will update *exec_attr(4)*, for the profile *Network Management*, as shown below.

Network Management:solaris:cmd:::/sbin/ipadm:euid=netadm;egid=netadm;privs=sys_ip_config, net_rawaccess

13 ipadm(1M) futures

As pointed out on the OpenSolaris mailing list[7] 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.

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

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-tuning 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 ID*s) per interface. Currently, each IP interface allows only one token to be set, i.e., once the command *create-addr -T addrconf* 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 *aobjname*, each corresponding to the *Interface ID* specified in the *create-addr* command.

14 Acknowledgments

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

James Carlson, Renee Danson, Nicolas Droux, Prasanna Kunisetty, Alan Maguire, Peter Memishian, Dan McDonald, Erik Nordmark, Darren Reed, Sebastien Roy, Rao Shoaib and Cathy Zhou.