IP Instances Interface Document

PSARC 2006/366

2006-12-28

Erik Nordmark
Yukun Zhang
Dong-Hai Han

1 Changes since commitment review

Based on the PSARC TCR at commitment review and code review comments the following changes have been made since commitment review:

- Removal of the -l option for zoneadm list (TCR)
- Addition of a zone property to dladm which can be displayed using dladm show-linkprop
- Renaming of the private zone_*_ifname() system calls to zone_*_datalink() to properly reflect that they are about datalink names
- Removal of dladmthe zone_get_ifnum() system call
- Addition of a “allow-exclusive-ip” boolean to the brand xml syntax in order to provide reasonable error reporting given that the lx brand can not currently support exclusive-IP zones (limitation in the lx brand ioctl emulation)

2 Introduction

IP instances introduces the ability to have separate zones connected to separate LANs or VLANs without introducing any network security issues. It does this by applying a simple approach to separate all the previously global variables in the TCP/IP collection of kernel modules, so that there can be multiple instances of them, and by providing some extensions to zonecfg to specify that a zone should have its own IP instance. In its simplest form what is needed in zonecfg is to add

```
set ip-type=exclusive
```

This addition to zonecfg is necessary because for compatibility reasons we do not believe we can change the existing behavior in zones networking, especially not in a Solaris 10 Update. Thus we can have a combination of “shared-IP zones” (today's S10 behavior) and the new “exclusive-IP zones” that are configured by setting the above zonecfg property.

An implication of having a separate IP instance per exclusive-IP zone is that all aspects of the TCP/IP state and policies become per IP instance. This includes the IP routing table. ARP table, IPsec policies and associations, pfhooks families and hooks, IP Filter rules, TCP/IP ndd variables. However, this has no effect on network protocols and services on top of TCP/UDP/IP; in particular, the behavior of NFS in zones is unchanged. To help clarify this we've renamed the project from “stack instances” to “IP Instances”.

Note that the requested release binding is micro/patch. This project is planning to integrate into S10U4.

3 Background – Networking and Zones

As customers are starting to deploy using Zones, some of them see a good fit was the design center for the Zones networking support; the servers to be consolidated are on the same IP subnet.

Other customers see some problems or limitations.

The problems stem from desiring to consolidate applications that need to communicate on different subnets that are on different VLANs or different LANs. The customer's we've talked that express the need for

- separate routing per zone
- prevent loopback traffic between zones inside the server

all have the expectation (and in some cases the explicit requirement) that the network traffic for a zone must be completely separate from the traffic for other zones; they might share the same physical NIC using VLANs, but that's the only allowed degree of sharing of the network.

Figure 1 shows a conceptual model of how Zones networking works in S10, with conceptually a separate TCP, UDP and SCTP for each zone, and incoming packets being demultiplexed to the separate transports based on their destination IP address. For outgoing packets, they end up with a source IP address which is one of the IP addresses assigned to the zone.

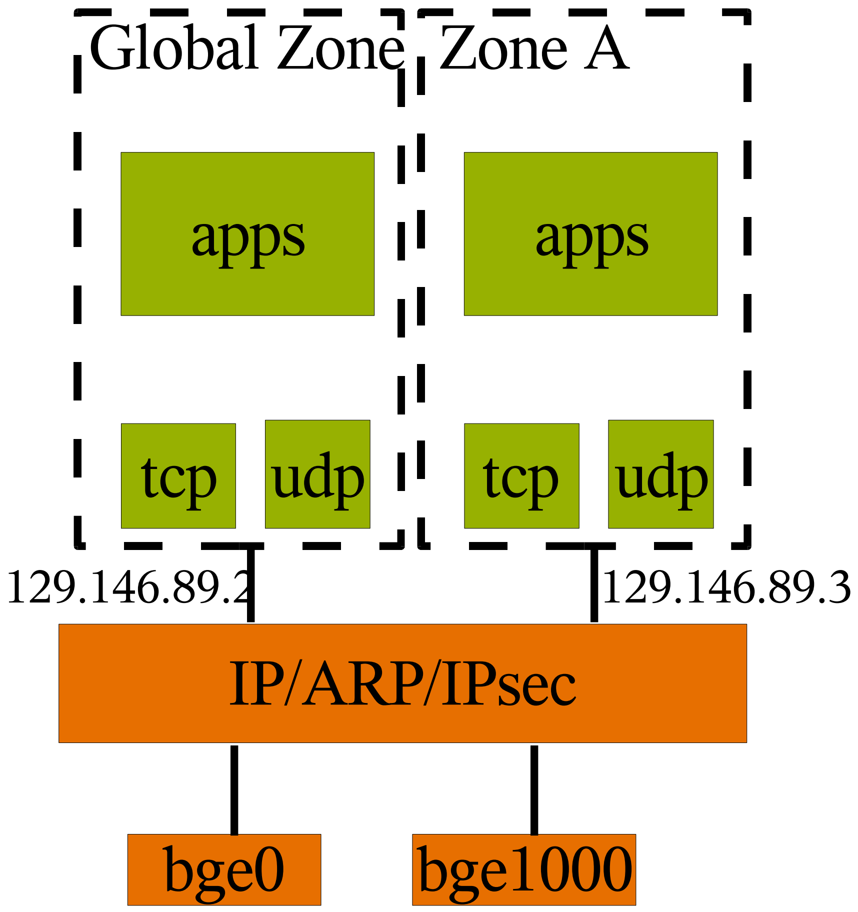

Figure 1: Conceptual model of zones networking

What zones provides for networking is name space isolation for the TCP/UDP/SCTP port name space by giving each zone its own distinct IP address(es). When the server is connected to multiple network interfaces, the shared IP behaves just as it behaved prior to the introduction of Zones; packets are routed as best viewed by IP. But what is needed when consolidating applications that need to be connected to isolated (V)LANs is in fact isolation for network connectivity. In the case of Solaris such isolation can be expressed in the form of which network interface names are accessible in each zone (since VLANs and other layer 2 concepts appear as separate network interface names). Then for each such zone there needs to be no internal IP-level sharing as depicted in figure 2.

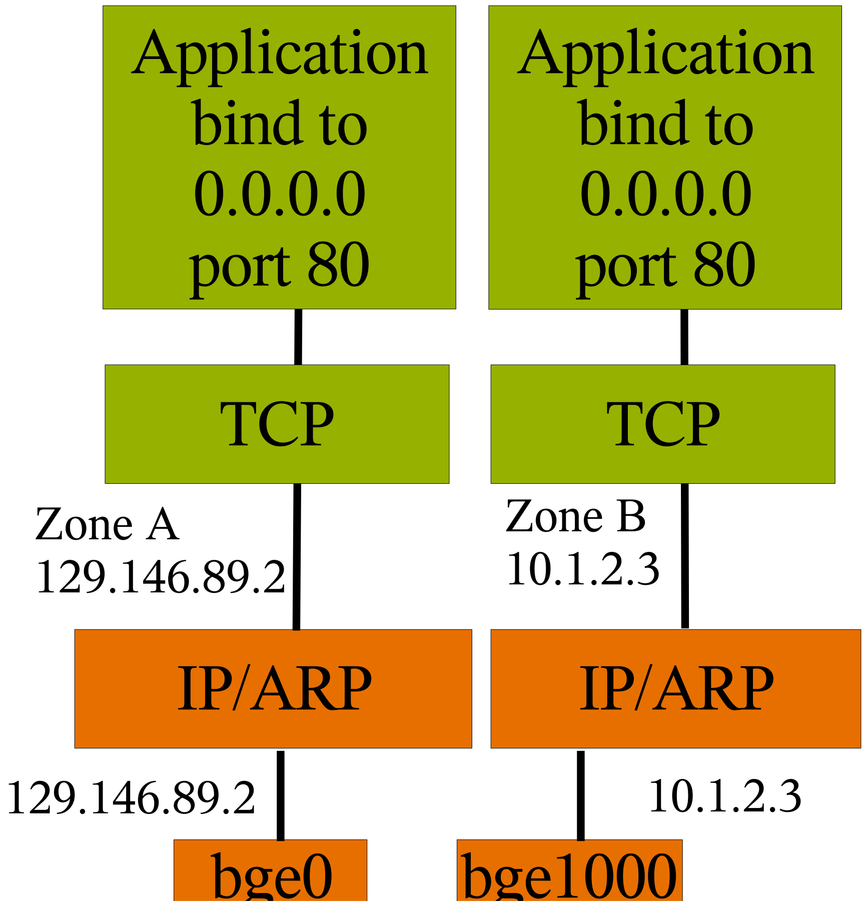

Figure 2: Conceptual model of zones networking with IP-level isolation

In many cases customers try to make their cases of different subnets and different (V)LANs work through some manual tweaking such as manually added routes and a global zone with IP address 0.0.0.0 on some NICs. But the Zones networking support was designed with a shared IP in mind and just separate IP address(es) per zone. Thus while Solaris ensures that a zone can only send IP packets with its IP addresses, and can only receive unicast packets address those IP addresses, there are separation issues when looking from the network in towards Solaris. Their implications are that:

- Even though some S10 RFEs were implemented to try to allow separate default routes per zone, this is hard to configure and doesn't work as expected. In some configurations it isn't possible to get working routing when separate default routes per zone are used.
- We can not guarantee that packets can not accidentally show up in a different zone that some security policy requires; there are just too many protocol features and code paths in IP, ARP, TCP, etc that can cause such "leakage" for packets arriving from the network.
- If/when the customer assumes that a zone can only send packets on the NICs on which they have IP addresses they become surprised. The design was to restrict the source addresses that a zone can use when sending packets, and the relationship between the NIC on which that IP address is configured and where packets get sent is quite weak and absent in some cases. (E.g., sending multicast or broadcast packets, sending ICMP errors and TCP resets, and sending packets after they have been queued waiting for ARP.)

- The behavior after the manual tweaking isn't guaranteed to work; we have no way to ensure that some customer invented approach or some blog suggestion will continue to work after a patch.
- The behavior with the manual tweaking is very fragile due to the “one bugfix at a time” evolution of the zones networking during S10 development. Today we are discovering unintended consequences of some of the code changes that were made so that different zones could use different default routes.

Note that some of the customers might express the issues they see as "it is too difficult to configure" a zone to have different default routes and get it to work, but the more fundamental issue seems to be that they expect secure separation (which is why they are using separate LANs or VLANs on the network), but they don't seem to see the security holes implicit in sharing the same IP and ARP for all the zones.

The result of this is that we are already starting to see escalations including escalated RFEs and these escalations can be extremely difficult to solve.

Part of the root cause for those is that Sun has and still do express networking aspects of zones as “separate networking” even though the separation is limited to the port number space. But with such vague descriptions combined with Zones overall being described as providing isolation, it isn't surprising if customers and the field assume that when zones are combined with network isolation (using separate LANs or VLANs), that the result will be isolation end-to-end from the network to the applications. In fact it takes a while to explain that even though the (V)LANs are separate and the applications are in separate zones, there is a shared IP in Solaris in between those whose role is to connect things together.

4 Justification

It is clear that a significant part of the customers that do application consolidation in the datacenter have a need to provide application isolation in combination with separate (V)LANs. Due to our unclear communication and the business needs, in some cases we end up having to address escalated RFEs. Thus one aspect of the business justification is that we must stop the bleeding and avoid wandering further into a support problem by providing an alternative to the S10 "shared IP" model. That translates into a desire to make this project available in the earliest possible S10 update.

In addition to providing the isolation in the short term, the architecture introduced by this project allows technological evolution for other parts networking. These future evolutions are not delivered by this project, but the added flexibility exemplified by them is part of the motivation for IP Instances. Examples are::

- Makes networking with zones look more similar to Xen, which means that we can leverage work (e.g., all the GLDv3 work) across Zones and Xen
- Easier testing of Solaris networking; can run multiple exclusive-IP zones inside one box and run automated testing that today requires several systems that are manually cabled together
- The potential to build more secure infrastructure by having a zone's sole connection to the network be an IPsec tunnel that is provided by the global zone
- The potential to leverage more network security pieces such as packet filtering capability for Zones as well as for Xen virtualization, by placing filtering hooks in the G:LD framework.
- The potential to have others leverage Solaris to build virtual network appliances where each appliance has an presence on the network which is independent of other appliances on the same box.

The BrandZ team also desires a supported way to run NAT in the global zone for a branded zone, so that only one IP address is needed for the system. Other approaches to providing NAT functionality in such configurations do not appear to be supportable due to the complex interdependencies that one would have to create inside the single IP instance. Thus the shortest time to market for such supported functionality is having an exclusive IP for the BrandZ zone, and have that communicate via the NAT in the global zone using an local-to-the box pseudo Ethernet (called vSwitch; something that is part of the Crossbow precut).

Note that in order to apply this approach to BrandZ, the branded zone has to do its own IP layer configuration (i.e., issue ifconfig or DHCP operations) since the global zone will not configure IP for a zone with an exclusive IP.

5 Goals

The goals of the project is to provide the option of complete IP separation for non-global zones while preserving the ability to have zones shared the same IP as the global zone. The kernel modules that will be modified to support IP instances is conceptually just IP. But since IP is made up of several components (such as IPsec, routing sockets, IP Filter), other code such as the transport protocols do direct function calls into the IP module, and IP depends on pfhooks(neti and hook) to provide the framework of packet filtering facility, the set of kernel modules expands to:

- ip, rts, arp, ipsec, ipsecesp, ipsecah, keysock, spdsock
- tcp, udp, sctp, icmp
- neti, hook, ipf

Specifying whether a shared or an exclusive IP is used will be done in zonecfg, to preserve the current approach to configuring networking for zones. The zonecfg specification for exclusive-IP zones will limit itself to specifying the boundary of the zone. For networking this means that zonecfg would just specify the set of network interfaces that the exclusive-IP zone has been granted.

The configuration of the internals of the exclusive-IP zone, such as whether static or dynamic IP addresses are used, netmasks, default routers, routing configuration in general, as well as the existing name service configuration, are done using existing Solaris methods.

Thus /etc/sysidcfg can be used to specify IP addresses and default routers just as for the global zone. Just as for the global zone, when the zone is booted the first time the sysidcfg is just to create /etc/hostname.<ifname> files, that the administrator can later modify. And just as for the global zone, if there is no sysidcfg when the exclusive-IP zone is booted the first time, sysidtool will run. For networking they will offer the set of network interface names that the global zone has assigned to the non-global zone.

Note that the DHCP client code and IPv6 stateless address autoconfiguration works unmodified in a zone with an exclusive IP, as well as several other IP-level capabilities that today are not available in a non-global zone.

The following particular items will work per exclusive-IP zone (without any specific changes – just from virtualizing the TCP/IP kernel state):

- DHCPv4 and IPv6 stateless address autoconfiguration
- IP Filter including its NAT functionality
- IPMP and MULTIRT
- IP routing

- ndd for setting TCP/UDP/SCTP as well as IP/ARP-level knobs
- IPsec/IKE

The network security of the resulting system should be at least as good as with separate Solaris servers connected to the same set of network. An exclusive-IP zone will be prevented from using network interfaces other than those that it has been explicitly assigned by the global zone. And the applications running in an exclusive-IP zone will have a limited access to the OS and hardware resources just as with zones today.

Additional security comes from preventing potential network attackers from using the shared IP in zones today to reach zones that shouldn't really be reachable from that network interface, or source route packets via the shared IP out some other network interface.

6 Non-goals

No virtualization of the IP Policy Framework and IPQoS. Those can still be used in the global zone to manage shared-IP zone's IP traffic, but their low usage and possible replacement by Crossbow network resource controls means there is no business justification for virtualization them.

No GLD or other layer 2 interface changes. This means that for instance it will not be possible for an exclusive-IP zone that has access to bge1001 and bge1002 to create an aggregation of those two; layer 2 aggregations must be created in the global zone. The project works with device drivers (current and future) that provide a /dev name for the interface e.g., /dev/bge0 today or /dev/net/bge3001 in the future (future names being provided by the Clearview project).

The project will not address any kernel components above the transport layer (such as NFS, KSSL, NL7C). Such components, when and if they are made zones aware, do not and should not be aware of IP instances; whether the IP addresses assigned to a zone is from an exclusive or shared IP is immaterial to components above the transport layer.

No change to how IP is configured in Solaris, in particular, this project does not take the various networking /etc files (/etc/hostname.<ifname>, /etc/defaultrouter, /etc/resolv.conf) and replace them by something else like SMF properties. A result of this is that for exclusive-IP zones we rely on the /etc files to configure the exclusive IP.

Due to a combination of the limited use of CGTP, the fact that the CGTP kernel components are not in the ON consolidation, and that both the CGTP kernel components and the CGTP hooks into IP would need to be revised in order to support correct CGTP filtering for exclusive-IP zones, the project will not revise the CGTP hooks. Thus CGTP will continue to work in the global zone. The base multi-routing support that CGTP uses will, just like all IP kernel components, work in exclusive-IP zones.

The IP instance concept only need to apply to kernel code that is "near" IP, because IP is the thing that is shared in the Solaris 10 zones model for networking. For instance, network applications are automatically separated for zones since each process runs in a zone. Furthermore, if kernel networking pieces that sit above UDP, TCP, SCTP, or RAWIP need to be virtualized for zones, they do not need to be aware of IP instances; such software can be virtualized by using the zoneid as the discriminator which applies whether or not IP is virtualized. Kernel software that is below IP, such a GLD or other network device drivers, doesn't have a need to be aware of IP instances either. Zones uses the /dev/ entry points as the way to control what each zone can access, and the device drivers already keep different instances (bge0 vs. bge1) apart. The only implication of this assumption is that is it sufficient to make the internal support routines (netstack framework) be project private.

We assume that there is no business need to make 3rd party software that sit between IP and the device drivers aware of zones and/or IP instances. This means that things like Firewall-1 and the Cisco VPN client can only be used in the shared-IP instance (used by the global zone.)

7 Technical components

There are seven primary technical components:

1. Introducing a model where a non-global zone can either share the IP instance with the global zone (which is the only possibility in S10), or have an exclusive-IP instance to itself. A zone with an exclusive-IP instance will need exclusive access to one or more network interfaces (could be separate LANs like bge1, or separate VLANs like bge2000; any datalink interface with a separate name in /dev/ can be used.)

   In this model, by design there is no sharing e.g., a zone with an exclusive IP has its own routing table, arp table, IPsec security policy and security associations, pfhooks families and hooks facilities, IP Filter rules and state, etc.

2. Extensions to zonecfg syntax to

   - Allow specifying that a zone has an exclusive-IP instance: 'set ip-type=exclusive'
   - For exclusive-IP zones, the net resource is limited to specifying only the physical property.

   Extensions to zoneadm list output format.

   Extensions to dladm to observe assignment of datalink names to zones, as well as being able to change that assignment for running zones.

3. Splitting the PRIV_SYS_NET_CONFIG privilege by introducing the new PRIV_SYS_IP_CONFIG privilege, that allows a subset of the operations allowed by PRIV_SYS_NET_CONFIG.

   A zone with an exclusive IP will be granted PRIV_SYS_IP_CONFIG and PRIV_SYS_NET_RAWACCES.

4. Changes to the networking SMF method scripts which today skip sections of the script if "zonename != global", to not skip those sections if the non-global zone has an exclusive IP.

5. Modifications to all the kernel modules that make up "IP" (such as /kernel/drv/ip and the IPsec modules), as well as all the kernel modules that perform function calls into IP (such as TCP and IP Filter), to allow multiple instances of all their writable global data.

6. A "netstack" kernel framework (netstack.h/netstack.c) which isolates the TCP/IP kernel modules from the mapping between zones and IP instances, and provides things like kstat_create_netstack() analogous to kstat_create_zone(). For further details on this, see below.

7. Virtualize the pfhooks framework and facilities to support a IP instances model. The pfhooks modules neti and hook have the idea of separate IP instances. Via having the pfhooks interfaces have one extra parameter of zoneid_t, and following, using one generic handle, we can support IP instances hooks families and hooks facilities to support instance specific IPFilter (or 3rd parties) firewalls.

8 Relationship between IP instances and zones

As depicted in figure 2, there can both be zones that use the shared IP, and there can be zones which have their own exclusive IP. Those two approaches are sufficient for our current needs. In the future, it might be possible to remove the use of the shared IP but this requires more carefully understanding whatever dependencies the administrator has on the current zones networking behavior.

The project has carefully tried to isolate the knowledge of the relationship between zones and IP instances so that, should there be need to solve different problems in this space in the future, we can easily build different relationships with minimal code change.

9 Configuration

The configuration aspects consists of extensions to zonecfg syntax to be able to configure zones with exclusive IPs. The configuration of IP-level aspects of the exclusive-IP zones are done using existing tools like sysidtool. Just like a sysidcfg can be deposited in the zone's /etc directory before the first boot in Solaris 10 (to specify timezone, root passwd, name service configuration ,etc), the same can be done for exclusive-IP zones. The difference is that for an exclusive-IP zone the sysidcfg can also specify IP-level configuration.

10 Zonecfg (committed)

The additions to zonecfg syntax are:

- New “ip-type” global property which can be set to “shared” (the default if not specified) or “exclusive”
- Only allowing a “physical” property with the “net” resource for exclusive-IP zones.

A result of these changes is that the output of zonecfg export changes.

The following examples shows how this changes things.

In S10s syntax we have something like

```
set zonepath=/export/zone1
add net
        set physical=bge1
        set address=10.1.2.3/24
end
```

If the global administrator wants zone1 be the only zone that uses bge1, then with IP instances the needed configuration will be:

```
set zonepath=/export/zone1
set ip-type=exclusive
add net
        set physical=bge1
end
```

with the green italic text above shows the additional pieces.

To prime the IP configuration the global zone administrator can create a sysidcfg in the /etc/ directory in the non-global zone containing:

```
network_interface=bge1
                {hostname=host_name
```

```
                        default_route=10.0.0.1
                        ip_address=10.1.2.3
                        netmask=255.0.0.0
                        protocol_ipv6=no}
```

Note that in S10 if the administrator tried to have bge1 be exclusively used by zone1 without IP instances, the admin would have had to create some scripts on her own to ifconfig plumb bge1 0.0.0.0 in the global zone and also add the default route using that script.

If the administrator instead wants to have the zone be configured using DHCP then what is specified with zonecfg would not change. But the above the sysidcfg would contain

```
     network_interface=bge1
                {dhcp protocol_ipv6=no}
```

And finally, if the global admin wants to have the non-global zone use dynamic address configuration for both IPv4 and IPv6, the sysidcfg would contain:

```
     network_interface=bge1
                {dhcp protocol_ipv6=yes}
```

11 zoneadm (committed)

There are some small interface changes for zoneadm. Some additional error messages are introduced for the case when an exclusive-IP zone have a network physical which is also assigned to some another running zone or used by the global zone.

For observability reasons there is a need for the global admin to see which running zones use an exclusive IP, and which datalink names those have been given. zoneadm list -v will be extended to have the IP type (and likewise, zoneadm -p will included it at the end of the line).

Thus zoneadm list -cv will generate something like:

```
  ID NAME             STATUS     PATH                           BRAND    IP
   0 global           running    /                              native  shared
   1 si               running    /export/si                     native  excl
   2 si-share         running    /export/si-share               native  shared
```

zoneadm list -p will generate something like:

```
0:global:running:/::native:shared
1:si:running:/export/si:a59aae4a-529c-40f1-d440-8cd29103d233:native:excl
2:si-share:running:/export/si-share:dd131889-0140-44f1-f954-
a599bad0fca0:native:shared
```

12 Dladm show-linkprop extension (committed)

In response to the TCR at commitment review, we have added a zone property to dladm.

This shows the zone a datalink name is currently assigned to. For instance,

```
#dadlm show-linkprop ath0
```

```
PROPERTY         VALUE           DEFAULT         POSSIBLE
channel          6               --              --
powermode        ?               off             off,fast,max
radio            ?               on              on,off
speed            54              --              1,2,5.5,6,9,11,12,18,24,36,48,54
zone             si              --              --
```

If the datalink has not been assigned to any zone, e.g., if it is being used in the global zone, the value will be displayed as '--'.

dladm can be used to temporarily change the assignment of datalinks to zones using dladm set-linkprop. (The persistent assignment is done using zonecfg.) Also, dladm reset-linkprop will "free" the datalink, that is, after a reset operation, the datalink belongs to no zone.

There are some limitations on these operations, the datalink must not be in use in other zones, including the global zone, or the set-linkprop will fail.

The set and reset operations work for GLDv3 datalinks only.

13 zonename (project private)

We have added a -t option to zonename that makes it print the ip-type thus it outputs either “shared” or “exclusive”. This is used by the SMF method scripts below. This is a project private interface. Should there in the future be a need to provide a committed interface, then it might make sense to introduce a new zoneprop command for this functionality.

14 SMF method scripts (project private)

The set of SMF services is unchanged, but for IP Filter the SMF methods appear in all zones as a result of the packaging changes.

Prior to IP instances a set of SMF method scripts (such as net-physical) had explicit checks for zonename == global, and in the case of a non-global zone the script would do very little if anything. Those 4 scripts are modified to instead check if the zone needs IP level configuration i.e. they check for zonename == global or zonename -t == shared. This is implemented by encapsulating it in a new smf routine (smf_configure_ip()) used by the other method scripts.

The remainder of the scripts are unchanged, that is, they continue to use /etc/hostname.<ifname> etc.

15 Netstack framework (project private)

The netstack framework provides a way for the TCP/IP kernel modules to be aware of each IP instance that is created and destroyed as part of zones being created and destroyed. Thus it avoids the need for each TCP/IP module to track which zones have exclusive IPs and which use the shared IP. It also provides interfaces (kstat_create_netstack()) that takes care of making the kstats available in the correct zones, for both the shared IP case and the exclusive IP case.

The netstack framework is implemented using zone_key_create(), kstat_create_zone(), kstat_zone_add(), etc. Thus it sits as a veneer between the TCP/IP kernel modules and the zones code providing the abstraction of a netstack_t.

In order to provide efficient linkage between different parts of the TCP/IP code, for instance TCP needing to call ire_route_lookup(), and since we don't need this framework to be extensible outside of the set of kernel modules which perform function calls into the TCP/IP code, the netstack framework is less general that the Zone zsd support in that each kernel module that uses it requires a #define of a moduleid in netstack.h and a recompile of the kernel.

The netstack functions provided to the rest of the TCP/IP code are:

```
void netstack_register(int moduleid,
 void *(*module_create)(netstackid_t, netstack_t *),
 void (*module_shutdown)(netstackid_t, void *),
 void (*module_destroy)(netstackid_t, void *));
void netstack_unregister(int moduleid);
```

The above functions are called from a modules _init() and _fini() functions.

The following functions are used e.g., by a TCP/IP module's open and close routines.

```
netstack_t *netstack_find_by_cred(const cred_t *);
netstack_t *netstack_find_by_stackid(netstackid_t);
void netstack_hold(netstack_t *);
void netstack_rele(netstack_t *);
zoneid_t netstackid_to_zoneid(netstackid_t);
netstackid_t zoneid_to_netstackid(zoneid_t);          [Consolidation private – see below]
```

The current netstack implementation has the netstackid as the same as the zoneid, but the implementation is structured so that the TCP/IP modules do not embed that knowledge but instead use the above two functions.

For the TCP/IP kstats we want to make them available to all the zones that use the same IP instance. That is the kstats for the shared IP should be visible in all the zones that use the shared IP, while the kstats for an exclusive IP should only be visible in the corresponding zone. The following two functions makes this trivial for the TCP/IP modules; they can just use kstat_create_netstack() instead of kstat_create() or kstat_create_zone():

```
kstat_t *kstat_create_netstack(char *, int, char *, char *, uchar_t,
 uint_t, uchar_t, netstackid_t);
void    kstat_delete_netstack(kstat_t *, netstackid_t);
```

In some cases, such as IPsec being loaded into the kernel, all the IP instances need to be notified. The following allows a module to walk all the netstacks:

```
typedef         int      netstack_handle_t;
void    netstack_next_init(netstack_handle_t *);
void    netstack_next_fini(netstack_handle_t *);
netstack_t      *netstack_next(netstack_handle_t *);
```

16 Privileges (committed)

The project introduces the new PRIV_SYS_IP_CONFIG as a subset of PRIV_SYS_NET_CONFIG. The definition of the two is:

privilege PRIV_SYS_IP_CONFIG

> Allows a process to configure a system's network interfaces and routes.
>
> Allows a process to configure TCP/IP network parameters using ndd.
>
> Allows a process access to otherwise restricted TCP/IP information using ndd.
>
> Allows a process to configure IPsec.
>
> Allows a process to pop anchored STREAMs modules with matching zoneid.

privilege PRIV_SYS_NET_CONFIG

> Allows all that PRIV_SYS_IP_CONFIG allows.
>
> Allows a process to push the rpcmod STREAMs module.
>
> Allows a process to INSERT/REMOVE STREAMs modules on locations other
>
> than the top of the module stack.

A zone with exclusive IP is granted PRIV_SYS_IP_CONFIG and PRIV_SYS_NET_RAW.

In addition, an exclusive-IP zone that is given a network interface (e.g., bge1) is given access to that /dev node (e.g., /dev/bge1). As a result, a process with sufficient privileges in the zone can snoop on bge1. But without access to e.g., /dev/bge0 the zone can not perform any operations (ifconfig plumb or snoop) on bge0.

17 Implications on Dynamic Reconfiguration (FYI)

There is no detailed specification for Solaris 10 on how Dynamic Reconfiguration of network interfaces is coordinated with the zones configuration. Some coordination is necessary between the DR operation and the zones' configuration, since the zones' configuration include the datalink name.

The following example shows a suggested sequence of operations that apply to S10 (the shared-IP) as well as exclusive-IP zones. The example assumes that the hardware platform supports hot-plug on network interface cards. In the example, the bge3 datalink is being removed and replaced with a e1000g7 datalink interface. In the case of the exclusive-IP zones we assume they all have VLAN interfaces on bge3.

1. The global administrator determines which zones will be affected by this reconfiguration.

   For the shared-IP setup this consists of reading the output of ifconfig -a to determine the set of running zones which have IP addresses on bge0:<N>

   For the exclusive-IP zones this consists of looking for bge.*003 in the output of dladm show-link.

   In both cases the global admin also has to inspect the configured but not running zones to see if they use bge3. For instance using zonecfg -z <zonename> info | grep bge3 for the shared-IP zones and grep for bge.*003 for the exclusive-IP zones.

2. For the installed but not running zones, zonecfg can be used to replace the physical with the new name. Thus for the shared-IP zones replace bge3 with e1000g7. And for the exclusive-IP zones, replace bge<N>003 with e1000g<N>007.
3. For the running zones, the non-global administrators have to be notified to coordinate the downtime.

   For the exclusive-IP zones the non-global admin needs to either be be told of the new network interface name, and manually update the IP configuration files (/etc/hostname.<ifname>, /etc/ipf/ipf.conf, etc.), or be told to do a sys-unconfig.

   For both the shared-IP and exclusive-IP zones, the non-global admin needs to check for any application configuration files which include the network interface name. For instance, IP multicast applications might be configured to know to use bge3.
4. The zones are halted. Zonecfg is used to replace physical=bge3 with physical=e1000g7. The DR out of bge3 is performed. The DR in of e1000g7 is performed. The zones are booted.

   For the case when sys-unconfig was done for an exclusive-IP zone in step 3, then the global admin can provide (perhaps with the help of the non-global admin), the /etc/sysidcfg content for the zone. If this is not done, then sysidtool will ask questions on the console.

Notice that for the shared-IP case it is possible to try to manually optimize the steps to not require a reboot. (With the global admin using ifconfig to move the configuration to the new network interface name.) But due to the concern, previously expressed by PSARC [6], about applications potentially embedding network interface names in their configuration, this seems undesirable to recommend to customers a DR approach which renames the network interfaces on a running system. Due to this concern this project has not explored ways to optimize the steps for exclusive-IP zones.

There are other examples of DR operations, such as combining DR with IPMP to handle network interfaces that are removed. But they all suffer from the lack (in S10; not introduced by this project) of any coordination between the datalink names available on the system and zonecfg.

The future Clearview vanity naming will certainly help to do DR of NICs with less of an impact.

18 platform.xml (Project Private)

The BrandZ project [4] introduced platform.xml, one for each supported zone type, to specify brand related informations, including the devices should appear in the zone. (Prior to that this knowledge was hard-coded in libzonecfg.)

With IP Instances, some new elements are added to platform.xml of native zone and sn1 zone, along with changes to the corresponding dtd file: zone_platform.dtd.1

For zone_platform.dtd.1, for definition of devices, a new attributes, ip-type is added, it is used to identify devices only available for certain IP types, which could be “shared” and “exclusive”, also it could be left blank, which means the device should appear in all kind of zones (both shared and exclusive).

For platform.xml in native and sn1 brand zones, a list of new “IP” devices are added, at this time, all of them are exclusive IP Instances only, for example:

```
<device match="ip" ip-type="exclusive" />

<device match="ip6" ip-type="exclusive" />
```

With this, /dev/ip and /dev/ip6 will appear in exclusive-IP zones, and exclusive-IP zones only.

Since the lx brand does not currently handle sufficient ioctl emulation to support exclusive-IP zone, we are also adding a “allow-exclusive-ip” boolean property to the platform.xml syntax.

For Solaris 10 patches/updates if BrandZ is not available, the IP Instances will be implemented using the libzonecfg hard-coded approach. Thus there is no dependency on the existence of platform.xml.

19 VLAN behavior changes (Not an interface)

With IP Instances, a VLAN provided by a GLDV3 DRIVER can be assigned to an exclusive-IP zone, and be used just like a physical link.

IP Instances doesn't introduce any new public interfaces for VLANs, but there are some implementation differences that affect what is seen in /dev of the zone. These are due to the need for a separate /dev/ entry for the datalink names that are delegated to a zone.

We already have e.g. /dev/bge<ppa> for some ppa, and the project extends that to cover VLAN ppas for the GLDv3 drivers, for example, some entry like /dev/bge33000 will appear in a zone, which corresponds to a VLAN on physical link bge0, with VLAN ID 33.

20 DLPI style-2 applications

There is potential issue in exclusive-IP zones for applications that *CAN ONLY* handle style-2, i.e. that will fail when then don't see /dev/bge (even though /dev/bge1 or /dev/bge33000) is on their zone. That issue is orthogonal to VLANs, since it appears for even bge1.

This limitation will be documented in the zones book.

21 Interactions with Trusted Extensions (FYI)

In Solaris 10 updates and Nevada there is support for Trusted Extensions, which use IP labels to determine which zones can be used. The trusted extensions can be configured in two different ways:

- Each zone has its own IP address(es), much like regular use of zones
- A single IP address is shared among all the zones, which means that the IP labels are used to determine to which zone a received packet should be delivered.

The introduction of IP instances makes it possible to consider extending TX to have a configuration where each label has not only its own IP address but also its own IP instance. The label usage will apply the same way as for the existing trusted configuration when each zone has its own IP address. This would potentially provide for better isolation when separate VLANs or separate LANs are used with TX.

The design, implementation, and testing of IP Instances enables such a configuration. However, the limitations of TX makes it impossible to run common configurations, like the trusted desktop, with IP-level isolation. This is because the communication with the X server in TX assume IP-level connectivity. Requiring such IP-level connectivity is at conflict with providing the higher-level security offered by IP Instances.

Various extensions to TX has been discussed to increase its security and make it able to operate with IP isolation:

- Enable the X-server to use pipes and/or AF_UNIX sockets to communicate between zones.
- Add TX label checks per network interface, and combine this with future Crossbow VNICs/vSwitches to have IP connectivity between the zones with strict label enforcement for those internal network interface names.

The TX team is exploring those options as part of understanding how TX can envolve from "labeled trust" to "labeled security". If they decide to use IP communication it would make sense to change the configurations to use IPv6 link-local addresses to avoid having to pick a unique IPv4 subnet number for the internal X communication.

22 Internal TCP/IP changes (implementation details)

The general structure of the changes to all the TCP/IP kernel modules consist of

- Defining a foo_stack_t that contains all the previously global data that is modified
- Having the init and fini routines call netstack_register() and netstack_unregister(), respectively
- Writing foo_stack_init() and foo_stack_fini() routines which allocate and initiate (and free) an instance of a foo_stack_t
- Making the per-instance (normally per-queue i.e. referenced by q_ptr) structure have a pointer to the foo_stack_t
- Making the foo_open() routine use netstack_find_by_cred() to get the pointer to the netstack_t and as a result a pointer to the foo_stack_t, so that this can be saved in the per-instance structure
- For modules that use kstat_create, modify them to use kstat_create_netstack() which ensures that the statistics are visible in the zones that use that IP instance.

All the TCP/IP kernel modules follow the above pattern.

23 Internal zones changes (project private)

The implementation of the zoneadm/zoneadmd changes result in introducing a new set of zones system calls. The details of these are all implementation specific. The list of new calls are:

```
extern int  zone_add_datalink(zoneid_t, char *);
extern int  zone_remove_datalink(zoneid_t, char *);
extern int  zone_check_datalink(zoneid_t *, char *);
extern int  zone_list_datalink(zoneid_t, int *, char *);
```

24 Extensions to the netinfo interfaces (consolidation private)

To support IP Filter for IP instances, both the neti and hook modules that were introduced by [3] are virtualized. Three consolidation private APIs changes to be zones and as a result, IP instances, aware.

We add a netstackid_t argument to net_register(), net_lookup(), and net_walk() to specify to which IP instance the caller is referring. The updated function are:

```
net_data_t net_register(const net_info_t *, netstackid_t);
net_data_t net_lookup(const char *, netstackid_t);
net_data_t net_walk(net_data_t *, netstackid_t);
```

The zoneid_to_netstackid() function is made consolidation private to have the same level as the above interfaces.

The other netinfo functions operation on a net_data_t handle, and the implementation of that handle is extended to track the IP instance.

As with BrandZ, IP Instances does not depend on [3], but the implementation is different whether or not it is present.

25 Internal pfhooks changes (project private)

As outlined above, the implementation of the net_data_t is extended. We add one member to the project private netd_netstack in struct net_data to point to the particular netstack_t. So all the other neti APIs, which use net_data as the handle, do not need the parameter zoneid_t.

```
struct net_data {
        LIST_ENTRY(net_data)            netd_list;
        net_info_t                      netd_info;
        int                             netd_refcnt;
        hook_family_int_t               *netd_hooks;
        netstack_t                      *netd_netstack;      <-- NEW
};
```

Internally pfhooks module neti and hook use the new added member netd_netstack in struct net_data as the indicator to specify the particular instance. So literally there are either no changes or one extra pointer to foo_stack_t in the neti and hook internal functions.

The taskq IP_INJECT_QUEUE_OUT and IP_INJECT_QUEUE_IN routines are made to be IP instances aware. ip_ni_queue_in_func() and ip_ni_queue_out_func(), which uses injection_t as the parameter, we add inj_ptr to tell them what foo_stack_t are generating the event.

```
typedef struct injection_s {
        net_inject_t    inj_data;
        boolean_t       inj_isv6;
        void *          inj_ptr;            <-- NEW
} injection_t;
```

For the internal interface hook_run(), we add one argument netstack_t pointer to tell what IP stack is checking the hook event.

```
extern int hook_run(hook_event_token_t, hook_data_t, netstack_t *);

                                                     NEW
```

26 References

[1] PSARC 2002/174 Virtualization and Namespace Isolation in Solaris

[2] PSARC 2002/188 Least privilege for Solaris

[3] PSARC 2005/334 Packet Filtering Hooks API

as modified by http://sac.eng/arc/PSARC/2005/334/opinion.ascii

[4] PSARC 2005/471 BrandZ: Support for non-native zones

[5] PSARC 2005/707 Surya: Forwarding Performance Enhancement

[6] PSARC 2004/702 GigaSwift Ethernet 2.0: Intel 1G Ethernet software