Datalink Administration from Non-Global Zones

Sebastien Roy

Project Clearview I-Team
clearview-discuss@opensolaris.org

Solaris Networking
Sun Microsystems, Inc.

Revision 1.0
July 23, 2009

Contents

1 Introduction

This case introduces architecture that lays the groundwork for the use of datalink management facilities from exclusive stack non-global zones. Shared-stack zones are out of scope since their networking model is restricted to the IP layer.

There is an existing need to use the `dladm(1M)` command within an exclusive stack non-global zone to show datalink configuration pertinent to that zone. This is documented in the following CR:

```
6509231 dladm should show links in exclusive stack zone
```

Further, some functionality that depends on `libdladm.so` APIs to browse datalink configuration simply won't work from exclusive stack non-global zones as shown here:

```
6835873 dlpi_walk() silently fails in an exclusive zone
```

The reason that this functionality doesn't work from non-global zones is simple. The framework that implements datalink management has no knowlege of zones or exclusive stacks. This framework includes the `dlmgmtd` daemon (accessed through a door that doesn't have a filesystem handle in non-global zones) and the GLDv3 kernel modules (accessed through a `/dev/dld` control device that doesn't exist in non-global zones).

This case defines the architecture that allows visibility into datalinks from non-global zones. In brief, all links assigned to a zone will be visible from that zone. For example, "`dladm show-link`" in an exclusive stack non-global zone will show the datalinks assigned to that zone.

In addition to the ability to view datalink configuration from non-global zones, architecture needs to be defined to meet the needs of projects that require the ability to create, modify, and delete datalink objects from non-global zones. Specifically, PSARC/2009/373 introduces IP tunnel datalinks, and requires that these datalinks be manageable from non-global zones.

This case defines the architecture needed for cases like PSARC/2009/373 to manage datalinks from non-global zones. It does not introduce the capability to manage any specific existing class of type of datalink from non-global zones, but defines the architecture necessary to make this possible.

2 Ten-Thousand Foot View

The general architecture for datalink management from non-global zones proposed by this design is one where each zone has its own persistent datalink configuration (`/etc/dladm/datalink.conf`), and all datalink ID management is done through a central `dlmgmtd` daemon running in the global zone. Each zone also has a `/dev/dld` device link so that libdladm functions can issue GLDv3 ioctls. The `/etc/dladm/datalink.conf` files in each zone contain the persistent link configuration for the links that are created in that zone.

An alternative design would be to have each zone run its own `dlmgmtd` daemon, but because the daemon manages the linkid space for the kernel, having multiple such processes would introduce a synchronization problem that is avoided entirely by maintaining a single `dlmgmtd` process. The `dlmgmtd` daemon can be viewed as an extension of the kernel itself[1].

[1]Note that the decision to have a user-land daemon manage the link-name to linkid mappings instead of the

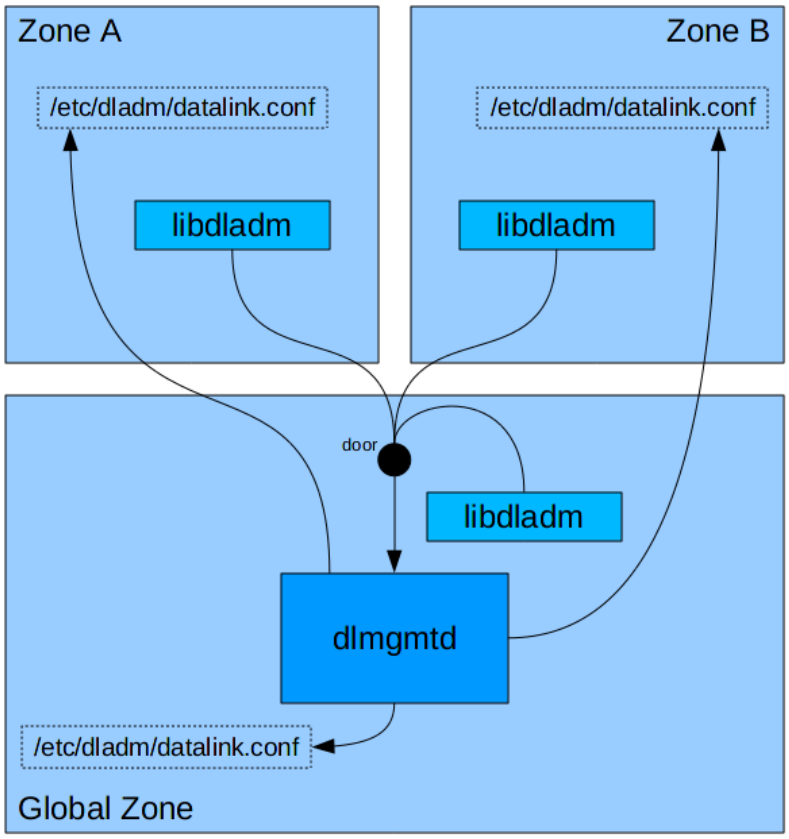

Figure 1: The global-zone dlmgmtd daemon is the arbiter of all linkid operations including reading and writing persistent configuration within each zone. The libdladm library calls in each zone communicate with dlmgmtd using door calls.

3 Per-Zone Datalink Namespace

Each zone needs its own unique namespace for datalink names. For example, the creation of a link named link0 in one zone shouldn't prevent another zone from creating a link of the same name. As a result, the dlmgmtd daemon (the entity that manages the linkid to name associations) will be modified to manage a link namespace that is effectively per-zone.

A given zone will only have visibility into the links that were created within that zone, and into links that are on-loan to that zone from the global zone (by having set its `zone` link property). This model preserves backward compatibility with the existing administrative model.

Note that today, link kstats (kstats for the "link" module named after the link name) have system-wide visibility. For example, if a system has a `vnic0` link, one can type `kstat -n vnic0` in any zone and get back the same data even if that link isn't assigned to the zone. While this is quite surprising and arguably a bug, it is squarely incompatible with the per-zone link namespace being

kernel was made by the Clearview UV component. The main reason for not maintaining these mappings in the kernel was that mappings exist for links that do not necessarily exist in the running kernel (links that were DR'ed out, or links that have persistent configuration that has not been applied).

proposed by this design. If the system has two `vnic0` links in two different zones, then there cannot be two kstats named `vnic0` with system-wide visibility. To solve this, link kstats will be modified to only have visibility in the zone where the link resides. For a link that is temporarily assigned to a non-global zone, a link kstat will exist in both the global zone and the non-global zone so that statistics tools keep working from both zones.

4 Access to the `dld` Control Device

The `libdladm.so` library operates on kernel datalink objects by issuing ioctls to the `/dev/dld` control device. In order for `dladm` to work correctly in a non-global zone, the `/dev/dld` device will be included by default in non-global zones' `/dev` directory.

5 Access to `dlmgmtd`

The `libdladm.so` library obtains linkid and persistent configuration information for datalinks by issuing door calls to the `dlmgmtd` door server accessed through the `/etc/svc/volatile/dladm/dlmgmt_door` door file.

The `dlmgmtd` daemon runs in the global zone, so in order for datalink management to be possible from a non-global zone, access to the `dlmgmtd` door server must be enabled for the non-global zone. To do this, the door file mentioned above will be created in each zone, but will be attached to the `dlmgmtd` door server running in the global zone.

This is done at zone boot time with the use of a new `DLMGMT_CMD_ZONEBOOT` `dlmgmtd` door call which is issued by `zoneadmd` (through a new `dladm_zone_boot()` `libdladm` function) when a zone boots. The door handler for this command in `dlmgmtd` is responsible for creating the temporary `<zone-root>/etc/svc/volatile/dladm/` directory and the included `dlmgmt_door` file, which is attached (using `fattach(3C)`) to the door server file descriptor. It then loads datalink information from the zone's `<zone-root>/etc/dladm/datalink.conf` into its internal datastructures.

The inverse is done when a zone is halted, using a new `dladm_zone_halt()` function and associated `DLMGMT_CMD_ZONEHALT` door command.

6 Access to `/etc/dladm/datalink.conf` in Non-Global Zones

Access to `/etc/dladm/datalink.conf` in non-global zones requires special care. The `dlmgmtd` daemon must protect the global zone's files and files in other non-global zones' filesystems from being pointed to by potential symbolic links in the non-global zone's `/etc/dladm/datalink.conf` path.

In order to write to the file in a safe and self-consistent manner, `dlmgmtd` first opens an alternate filename for writing and writes to that file. If successful, it then renames this alternate file to the `/etc/dladm/datalink.conf` location. If an error occurs while writing to the alternate file, it unlinks this alternate file leaving the original `/etc/dladm/datalink.conf` intact. There are thus three required operations on files in non-global zones:

1. Open a file
2. Rename a file

3. Unlink a file

All of these need to be performed in the context of the non-global zone. This is accomplished by forking a child which does a `zone_enter()` in order to perform each operation. In the case of opening a file, the file-descriptor opened by the child in the non-global zone must be passed back to the parent so that the actual writing can occur. This is done by having the child pass the file-descriptor back to the parent through a pipe using the `I_SENDFD` ioctl. The same trick is used to allow the parent to open the file for reading.

7 Privilege Model for Non-Global Zones

This case depends on privileges (or a zone's lack thereof) to restrict access to functionality that is not yet zone-aware. The `sys_dl_config` privilege is currently required to make modifications to any link configuration. This privilege is not available from non-global zones by design, and this design does not require that this privilege will ever be needed from non-global zones. As support for the managment of a particular link class is added, a privilege should be defined as a requirement for modification of such link objects, and this privilege should be delegated to non-global zones. The `sys_dl_config` privilege would encompass this finer-grained privilege.

For example, a future case that adds support for VNIC management in non-global zones would also introduce a `sys_vnic_config` privilege that would be delegated to non-global zones. The broader `sys_dl_config`, on the other hand, would not. As a more specific example, PSARC/2009/373 defines the `sys_iptun_config` privilege that is required to manage IP tunnel links, and that privilege will be delegated to non-global zones.

This case does not, however, define any such privilege because it requires future cases to introduce the functionality that such privileges will be associated with.

8 Zone-ID Filtering of GLDv3 Operations

Restricting the set of datalink-related privileges delegated to non-global zones restricts the types of modifications that non-global zone processes can make to the system, but it does nothing about the type of information that is visible using unprivileged operations.

The operations performed by the `dladm show-*` subcommands currently don't require privileges, and this architecture does not change that. These subcommands issue ioctls that are processed by various GLDv3 kernel modules, and these ioctls are registered with the GLDv3 framework by each kernel module using the `dld_ioc_register()` function.

The various show subcommands will remain unprivileged, but the kernel will be modified so that link information from one zone is not leaked into another. For example, `show-vnic` will only show VNICs assigned to the caller's zone, and similarly for other show subcommands.

Some show subcommands such as `show-link`, `show-vnic`, `show-vlan`, and `show-aggr` display link names of underlying links in the `OVER` column. In cases where these underlying links don't belong to the zone of the caller, the `dladm` command won't be able to obtain those link names, and will instead print a '?' character denoting that the information is unknown. For example:

```
bash-3.2# zonename
test
bash-3.2# dladm show-link
```

```
LINK        CLASS    MTU    STATE    OVER
vnic0       vnic     1500   up       ?
bash-3.2# dladm show-vnic
LINK         OVER         SPEED  MACADDRESS        MACADDRTYPE         VID
vnic0        ?            1000   2:8:20:6e:da:4d   random              0
```

In the example above, the link `vnic0` is assigned to a non-global zone named `test`. The VNIC is configured over a link named `bge1` which belongs to the global zone, and that link name is omitted from the output.

9 Automatic Deletion of Links at Zone Shutdown

The creation of datalinks in non-global zones poses an interesting problem that occurs when a zone halts. Specifically, if a link is created while the zone was running, that link must not exist in the kernel after the zone has ceased to run. When the zone halts, something must iterate through the list of such links and temporarily delete them.

One option is to have `zoneadmd` iterate over these links and issue the appropriate `libdladm` functions to delete them. This would introduce complexity in `zoneadmd` as it would require it to know how to delete each class of datalink (there are different functions responsible for for each class).

The above approach is also not in-line with how IP interfaces created within a zone are automatically deleted when the zone halts. Such interfaces are automatically deleted in the kernel when the stack shuts down using netstack callbacks registered by `netstack_register()`.

The design chosen is in-line with how IP interfaces are automatically deleted. The `dls` module and modules responsible for the kernel datalink objects will each have an entry at the very top of `netstack_t`, in front of all other modules, as shown in the below excerpt from `<sys/netstack.h>`. In this example, a hypothetical `foo` module is responsible for the configuration of links of class `foo`, and is thus responsible for the deletion of this class of link during zone halt. Future cases introducing zones support for a specific class of datalink (e.g. `vnic`, or `iptun` as specified in PSARC/2009/373) would use `foo` as an analogy.

```
/*
 * One for each module which uses netstack support.
 * Used in netstack_register().
 *
 * The order of these is important for some modules both for
 * the creation (which done in ascending order) and destruction (which is
 * done ine in decending order).
 */
#define NS_ALL          -1      /* Match all */
#define NS_DLS          0
#define NS_FOO          1
#define NS_STR          2       /* autopush list etc */
#define NS_HOOK         3
#define NS_NETI         4
#define NS_ARP          5
#define NS_IP           6
...
```

As noted in the comment above, the order of the above modules is critical because IP interfaces

over a given link have to be destroyed before that link is destroyed. Below is the order of events at zone shutdown as it relates to the modules relevant to the discussion, `NS_IP`, `NS_STR`, `NS_FOO`, and `NS_DLS` (note that there are two stages to a zone's shutdown: shutdown, and halt)

1. `NS_IP` `ip_stack_shutdown()` is called. This removes loopback IP interfaces and does other housekeeping to prepare for halting.
2. `NS_STR` `str_stack_shutdown` is called. This unplumbs (using `I_PUNLINK`) all IP interfaces in the given stack.
3. `NS_FOO` `foo_stack_shutdown()` is called. This deletes all `foo` links created within the zone. Note that this specifically does not delete `foo` links that were assigned to the zone from the global zone. Such links should not be deleted, but merely assigned back to the global zone. This is done later by the `NS_DLS` `dls_stack_shutdown()` callback.
4. `NS_DLS` `dls_stack_shutdown()` is called. This removes all links remaining in the zone from the zone by assigning them back to the global zone. Note that this must be done after other callbacks have destroyed IP interfaces in the zone, and links that were created within the zone. The `dls` module is optimally positioned to do this task since this is a generic operation which is not specific to any class of link. It is also the module responsible for assigning links to zones when their `zone` property is set.

   Also note that this link zone reassignment was previously done by `zoneadmd` in its `unconfigure_exclusive_network_interfaces_run()` function. Moving this out of `zoneadmd` and into the kernel netstack infrastructure makes the architecture and code cleaner, and makes the order of operations explicit in the way modules are positioned in the `netstack_t`.
5. `NS_IP` `ip_stack_fini()` is called. This frees the ip module's stack-local `ip_stack_t` and datastructures therein.
6. `NS_STR` `str_stack_fini()` is called. This frees the `str_stack_t`.
7. `NS_FOO` `foo_stack_fini()` is called. This frees the `foo_stack_t`.
8. `NS_DLS` `dls_stack_fini()` is called, which frees the `dls_stack_t` for this stack.

10 Zone Migration Considerations

A subtle a new problem is posed by having per-zone `/etc/dladm/datalink.conf` files. The link ID for each datalink is stored in that file, yet the linkid space global to the system (not per-zone). When a zone is migrated from one system to another, that file goes along with it, and those link IDs are meaningless on the target system. There may be existing links on the target system with those link IDs, so they cannot be used.

The solution chosen is to not store link IDs in `/etc/dladm/datalink.conf`. When the file is loaded, each link is assigned a unique link ID that is only valid in the running system. References to other links in the file (for example for VLANs, VNICs, or aggregations over other links) are done by name instead of by link ID. Since the link namespace is per-zone, this scheme is compatible with zone migration.

A Future Work Related to Datalink Administration in Zones

The design choices made here also allow for future work related to datalink administration and zones. For example, in addition to PSARC/2009/373 which introduces administration of IP tunnel

links and their administration from non-global zones, the following could be possible:

- Administration of VNICs from within non-global zones.

  The framework will be in place to allow a potential future projects to add more administrative capabilities to the kernel. Administration of VNICs is one example. Perhaps there could be value in allowing a non-global zone to create VNICs over a physical link that is assigned to the zone.

  This could also be coupled with per-zone network resource management configured using zonecfg such as bandwidth limits for such VNICs.

- The all-seeing global zone.

  As mentioned in section 3, the link namespace model being implemented by this design maintains backward compatibility, but more possibilities exist for the future. For example, the global zone could have visibility into all links in all zones, even those that were created in non-global zones. If the creation of VNICs from non-global zones is ever implemented, this namespace model might be more attractive, as VNICs use system resources, and giving them global visibility would be beneficial.

  Because the link namespace would still be per-zone, the global zone's administrative tools would then need to differentiate the links per-zone, perhaps by prefixing link names with a zone-name identifier (e.g., webzone:vnic0).

- Automatic creation and assignment of VNICs for exclusive-stack zones.

  Instead of requiring an administrator to create a VNIC which is then assigned to a non-global zone using zonecfg, one can envision a model where the zones framework is told to automatically create a VNIC over a given physical interface on the fly for a zone when it boots, and assigns that VNIC to the zone.

- Administration of flows from non-global zones.

  The usage of the `flowadm(1M)` and upcoming `flowstat(1M)` commands from non-global zones are not addressed by this design and are potential for future work.