SCSI FC Tansport

Date: 11/18/2008

Kernel Revisions for features:

```
rports : <<TBS>>
vports : 2.6.22
bsg support : 2.6.30 (?TBD?)
```

Introduction

This file documents the features and components of the SCSI FC Transport. It also provides documents the API between the transport and FC LLDDs.

The FC transport can be found at:

```
drivers/scsi/scsi_transport_fc.c
include/scsi/scsi_transport_fc.h
include/scsi/scsi_netlink_fc.h
include/scsi/scsi_bsg_fc.h
```

This file is found at Documentation/scsi/scsi_fc_transport.rst

FC Remote Ports (rports)

<< To Be Supplied >>

FC Virtual Ports (vports)

Overview

New FC standards have defined mechanisms which allows for a single physical port to appear on as multiple communication ports. Using the N_Port Id Virtualization (NPIV) mechanism, a point-to-point connection to a Fabric can be assigned more than 1 N_Port_ID. Each N_Port_ID appears as a separate port to other endpoints on the fabric, even though it shares one physical link to the switch for communication. Each N_Port_ID can have a unique view of the fabric based on fabric zoning and array lun-masking (just like a normal non-NPIV adapter). Using the Virtual Fabric (VF) mechanism, adding a fabric header to each frame allows the port to interact with the Fabric Port to join multiple fabrics. The port will obtain an N_Port_ID on each fabric it joins. Each fabric will have its own unique view of endpoints and configuration parameters. NPIV may be used together with VF so that the port can obtain multiple N_Port_IDs on each virtual fabric.

The FC transport is now recognizing a new object - a vport. A vport is an entity that has a world-wide unique World Wide Port Name (wwpn) and World Wide Node Name (wwnn). The transport also allows for the FC4's to be specified for the vport, with FCP_Initiator being the primary role expected. Once instantiated by one of the above methods, it will have a distinct N_Port_ID and view of fabric endpoints and storage entities. The fc_host associated with the physical adapter will export the ability to create vports. The transport will create the vport object within the Linux device tree, and instruct the fc_host's driver to instantiate the virtual port. Typically, the driver will create a new scsi_host instance on the vport, resulting in a unique <H,C,T,L> namespace for the vport. Thus, whether a FC port is based on a physical port or on a virtual port, each will appear as a unique scsi_host with its own target and lun space.

> **Note**
>
> At this time, the transport is written to create only NPIV-based vports. However, consideration was given to VF-based vports and it should be a minor change to add support if needed. The remaining discussion will concentrate on NPIV.

> **Note**
>
> World Wide Name assignment (and uniqueness guarantees) are left up to an administrative entity controlling the vport. For example, if vports are to be associated with virtual machines, a XEN mgmt utility would be responsible for creating wwpn/wwnn's for the vport, using its own naming authority and OUI. (Note: it already does this for virtual MAC addresses).

Device Trees and Vport Objects:

Today, the device tree typically contains the scsi_host object, with rports and scsi target objects underneath it. Currently

the FC transport creates the vport object and places it under the scsi_host object corresponding to the physical adapter. The LLDD will allocate a new scsi_host for the vport and link its object under the vport. The remainder of the tree under the vports scsi_host is the same as the non-NPIV case. The transport is written currently to easily allow the parent of the vport to be something other than the scsi_host. This could be used in the future to link the object onto a vm-specific device tree. If the vport's parent is not the physical port's scsi_host, a symbolic link to the vport object will be placed in the physical port's scsi_host.

Here's what to expect in the device tree :

The typical Physical Port's Scsi_Host:

```
/sys/devices/.../host17/
```

and it has the typical descendant tree:

```
/sys/devices/.../host17/rport-17:0-0/target17:0:0/17:0:0:0:
```

and then the vport is created on the Physical Port:

```
/sys/devices/.../host17/vport-17:0-0
```

and the vport's Scsi_Host is then created:

```
/sys/devices/.../host17/vport-17:0-0/host18
```

and then the rest of the tree progresses, such as:

```
/sys/devices/.../host17/vport-17:0-0/host18/rport-18:0-0/target18:0:0/18:0:0:0:
```

Here's what to expect in the sysfs tree:

```
scsi_hosts:
  /sys/class/scsi_host/host17               physical port's scsi_host
  /sys/class/scsi_host/host18               vport's scsi_host
fc_hosts:
  /sys/class/fc_host/host17                 physical port's fc_host
  /sys/class/fc_host/host18                 vport's fc_host
fc_vports:
  /sys/class/fc_vports/vport-17:0-0         the vport's fc_vport
fc_rports:
  /sys/class/fc_remote_ports/rport-17:0-0   rport on the physical port
  /sys/class/fc_remote_ports/rport-18:0-0   rport on the vport
```

Vport Attributes

The new fc_vport class object has the following attributes

node_name: Read_Only
: The WWNN of the vport

port_name: Read_Only
: The WWPN of the vport

roles: Read_Only
: Indicates the FC4 roles enabled on the vport.

symbolic_name: Read_Write
: A string, appended to the driver's symbolic port name string, which is registered with the switch to identify the vport. For example, a hypervisor could set this string to "Xen Domain 2 VM 5 Vport 2", and this set of identifiers can be seen on switch management screens to identify the port.

vport_delete: Write_Only
: When written with a "1", will tear down the vport.

vport_disable: Write_Only
: When written with a "1", will transition the vport to a disabled. state. The vport will still be instantiated with the Linux kernel, but it will not be active on the FC link. When written with a "0", will enable the vport.

vport_last_state: Read_Only
: Indicates the previous state of the vport. See the section below on "Vport States".

vport_state: Read_Only
: Indicates the state of the vport. See the section below on "Vport States".

vport_type: Read_Only
: Reflects the FC mechanism used to create the virtual port. Only NPIV is supported currently.

For the fc_host class object, the following attributes are added for vports:

max_npiv_vports: Read_Only

Indicates the maximum number of NPIV-based vports that the driver/adapter can support on the fc_host.

npiv_vports_inuse: Read_Only

Indicates how many NPIV-based vports have been instantiated on the fc_host.

vport_create: Write_Only

A "simple" create interface to instantiate a vport on an fc_host. A "<WWPN>:<WWNN>" string is written to the attribute. The transport then instantiates the vport object and calls the LLDD to create the vport with the role of FCP_Initiator. Each WWN is specified as 16 hex characters and may *not* contain any prefixes (e.g. 0x, x, etc).

vport_delete: Write_Only

A "simple" delete interface to teardown a vport. A "<WWPN>:<WWNN>" string is written to the attribute. The transport will locate the vport on the fc_host with the same WWNs and tear it down. Each WWN is specified as 16 hex characters and may *not* contain any prefixes (e.g. 0x, x, etc).

Vport States

Vport instantiation consists of two parts:

- Creation with the kernel and LLDD. This means all transport and driver data structures are built up, and device objects created. This is equivalent to a driver "attach" on an adapter, which is independent of the adapter's link state.
- Instantiation of the vport on the FC link via ELS traffic, etc. This is equivalent to a "link up" and successful link initialization.

Further information can be found in the interfaces section below for Vport Creation.

Once a vport has been instantiated with the kernel/LLDD, a vport state can be reported via the sysfs attribute. The following states exist:

FC_VPORT_UNKNOWN - Unknown

An temporary state, typically set only while the vport is being instantiated with the kernel and LLDD.

FC_VPORT_ACTIVE - Active

The vport has been successfully been created on the FC link. It is fully functional.

FC_VPORT_DISABLED - Disabled

The vport instantiated, but "disabled". The vport is not instantiated on the FC link. This is equivalent to a physical port with the link "down".

FC_VPORT_LINKDOWN - Linkdown

The vport is not operational as the physical link is not operational.

FC_VPORT_INITIALIZING - Initializing

The vport is in the process of instantiating on the FC link. The LLDD will set this state just prior to starting the ELS traffic to create the vport. This state will persist until the vport is successfully created (state becomes FC_VPORT_ACTIVE) or it fails (state is one of the values below). As this state is transitory, it will not be preserved in the "vport_last_state".

FC_VPORT_NO_FABRIC_SUPP - No Fabric Support

The vport is not operational. One of the following conditions were encountered:

- The FC topology is not Point-to-Point
- The FC port is not connected to an F_Port
- The F_Port has indicated that NPIV is not supported.

FC_VPORT_NO_FABRIC_RSCS - No Fabric Resources

The vport is not operational. The Fabric failed FDISC with a status indicating that it does not have sufficient resources to complete the operation.

FC_VPORT_FABRIC_LOGOUT - Fabric Logout

The vport is not operational. The Fabric has LOGO'd the N_Port_ID associated with the vport.

FC_VPORT_FABRIC_REJ_WWN - Fabric Rejected WWN

The vport is not operational. The Fabric failed FDISC with a status indicating that the WWN's are not valid.

FC_VPORT_FAILED - VPort Failed

The vport is not operational. This is a catchall for all other error conditions.

The following state table indicates the different state transitions:

State	Event	New State
n/a	Initialization	Unknown
Unknown:	Link Down	Linkdown
	Link Up & Loop	No Fabric Support
	Link Up & no Fabric	No Fabric Support
	Link Up & FLOGI response indicates no NPIV support	No Fabric Support
	Link Up & FDISC being sent	Initializing
	Disable request	Disable
Linkdown:	Link Up	Unknown
Initializing:	FDISC ACC	Active
	FDISC LS_RJT w/ no resources	No Fabric Resources
	FDISC LS_RJT w/ invalid pname or invalid nport_id	Fabric Rejected WWN
	FDISC LS_RJT failed for other reasons	Vport Failed
	Link Down	Linkdown
	Disable request	Disable
Disable:	Enable request	Unknown
Active:	LOGO received from fabric	Fabric Logout
	Link Down	Linkdown
	Disable request	Disable
Fabric Logout:	Link still up	Unknown

The following 4 error states all have the same transitions:

```
No Fabric Support:
No Fabric Resources:
Fabric Rejected WWN:
Vport Failed:
                    Disable request                 Disable
                    Link goes down                  Linkdown
```

Transport <-> LLDD Interfaces

Vport support by LLDD:

The LLDD indicates support for vports by supplying a vport_create() function in the transport template. The presence of this function will cause the creation of the new attributes on the fc_host. As part of the physical port completing its initialization relative to the transport, it should set the max_npiv_vports attribute to indicate the maximum number of vports the driver and/or adapter supports.

Vport Creation:

The LLDD vport_create() syntax is:

```
int vport_create(struct fc_vport *vport, bool disable)
```

where:

vport	Is the newly allocated vport object
disable	If "true", the vport is to be created in a disabled stated. If "false", the vport is to be enabled upon creation.

When a request is made to create a new vport (via sgio/netlink, or the vport_create fc_host attribute), the transport will validate that the LLDD can support another vport (e.g. max_npiv_vports > npiv_vports_inuse). If not, the create request will be failed. If space remains, the transport will increment the vport count, create the vport object, and then call the LLDD's vport_create() function with the newly allocated vport object.

As mentioned above, vport creation is divided into two parts:

- Creation with the kernel and LLDD. This means all transport and driver data structures are built up, and device objects created. This is equivalent to a driver "attach" on an adapter, which is independent of the adapter's link state.
- Instantiation of the vport on the FC link via ELS traffic, etc. This is equivalent to a "link up" and successful link initialization.

The LLDD's vport_create() function will not synchronously wait for both parts to be fully completed before returning. It must validate that the infrastructure exists to support NPIV, and complete the first part of vport creation (data structure build up) before returning. We do not hinge vport_create() on the link-side operation mainly because:

- The link may be down. It is not a failure if it is. It simply means the vport is in an inoperable state until the link comes up. This is consistent with the link bouncing post vport creation.
- The vport may be created in a disabled state.
- This is consistent with a model where: the vport equates to a FC adapter. The vport_create is synonymous with driver attachment to the adapter, which is independent of link state.

> **Note**
>
> special error codes have been defined to delineate infrastructure failure cases for quicker resolution.

The expected behavior for the LLDD's vport_create() function is:

- Validate Infrastructure:
  - If the driver or adapter cannot support another vport, whether due to improper firmware, (a lie about) max_npiv, or a lack of some other resource - return VPCERR_UNSUPPORTED.
  - If the driver validates the WWN's against those already active on the adapter and detects an overlap - return VPCERR_BAD_WWN.
  - If the driver detects the topology is loop, non-fabric, or the FLOGI did not support NPIV - return VPCERR_NO_FABRIC_SUPP.
- Allocate data structures. If errors are encountered, such as out of memory conditions, return the respective negative Exxx error code.
- If the role is FCP Initiator, the LLDD is to :
  - Call scsi_host_alloc() to allocate a scsi_host for the vport.
  - Call scsi_add_host(new_shost, &vport->dev) to start the scsi_host and bind it as a child of the vport device.
  - Initializes the fc_host attribute values.
- Kick of further vport state transitions based on the disable flag and link state - and return success (zero).

LLDD Implementers Notes:

- It is suggested that there be a different fc_function_templates for the physical port and the virtual port. The physical port's template would have the vport_create, vport_delete, and vport_disable functions, while the vports would not.
- It is suggested that there be different scsi_host_templates for the physical port and virtual port. Likely, there are driver attributes, embedded into the scsi_host_template, that are applicable for the physical port only (link speed, topology setting, etc). This ensures that the attributes are applicable to the respective scsi_host.

Vport Disable/Enable:

The LLDD vport_disable() syntax is:

```
int vport_disable(struct fc_vport *vport, bool disable)
```

where:

vport	Is vport to be enabled or disabled
disable	If "true", the vport is to be disabled. If "false", the vport is to be enabled.

When a request is made to change the disabled state on a vport, the transport will validate the request against the existing vport state. If the request is to disable and the vport is already disabled, the request will fail. Similarly, if the request is to enable, and the vport is not in a disabled state, the request will fail. If the request is valid for the vport state, the transport will call the LLDD to change the vport's state.

Within the LLDD, if a vport is disabled, it remains instantiated with the kernel and LLDD, but it is not active or visible on the FC link in any way. (see Vport Creation and the 2 part instantiation discussion). The vport will remain in this state until it is deleted or re-enabled. When enabling a vport, the LLDD reinstantiates the vport on the FC link - essentially restarting the LLDD statemachine (see Vport States above).

Vport Deletion:

The LLDD vport_delete() syntax is:

```
int vport_delete(struct fc_vport *vport)
```

where:

vport: Is vport to delete

When a request is made to delete a vport (via sgio/netlink, or via the fc_host or fc_vport vport_delete attributes), the transport will call the LLDD to terminate the vport on the FC link, and teardown all other datastructures and references. If the LLDD completes successfully, the transport will teardown the vport objects and complete the vport removal. If the LLDD delete request fails, the vport object will remain, but will be in an indeterminate state.

Within the LLDD, the normal code paths for a scsi_host teardown should be followed. E.g. If the vport has a FCP Initiator role, the LLDD will call fc_remove_host() for the vports scsi_host, followed by scsi_remove_host() and scsi_host_put() for the vports scsi_host.

Other:

fc_host port_type attribute:

There is a new fc_host port_type value - FC_PORTTYPE_NPIV. This value must be set on all vport-based fc_hosts. Normally, on a physical port, the port_type attribute would be set to NPORT, NLPORT, etc based on the topology type and existence of the fabric. As this is not applicable to a vport, it makes more sense to report the FC mechanism used to create the vport.

Driver unload:

FC drivers are required to call fc_remove_host() prior to calling scsi_remove_host(). This allows the fc_host to tear down all remote ports prior the scsi_host being torn down. The fc_remove_host() call was updated to remove all vports for the fc_host as well.

Transport supplied functions

The following functions are supplied by the FC-transport for use by LLDs.

fc_vport_create	create a vport
fc_vport_terminate	detach and remove a vport

Details:

```
/**
* fc_vport_create - Admin App or LLDD requests creation of a vport
* @shost:     scsi host the virtual port is connected to.
* @ids:       The world wide names, FC4 port roles, etc for
*              the virtual port.
*
* Notes:
*     This routine assumes no locks are held on entry.
*/
struct fc_vport *
fc_vport_create(struct Scsi_Host *shost, struct fc_vport_identifiers *ids)

/**
* fc_vport_terminate - Admin App or LLDD requests termination of a vport
* @vport:      fc_vport to be terminated
*
* Calls the LLDD vport_delete() function, then deallocates and removes
* the vport from the shost and object tree.
*
* Notes:
*      This routine assumes no locks are held on entry.
*/
int
fc_vport_terminate(struct fc_vport *vport)
```

FC BSG support (CT & ELS passthru, and more)

<< To Be Supplied >>

Credits

The following people have contributed to this document:

James Smart james.smart@emulex.com