# NVMe Over Fabrics in FreeBSD

BY JOHN BALDWIN

**NVM Express (NVMe)** is a recent standard providing access to non-volatile memory block storage devices such as SSDs. NVMe was originally defined to access non-volatile memory devices via PCI-express. This includes register definitions for the PCI-express controller device, the layout and structure of command submission and completion queues stored in main memory, and sets of commands.

The base NVMe specification defines an **Admin Command Set** used on a single admin submission and completion queue pair associated with each controller. Administrative commands do not handle I/O requests. Instead, these commands are used to create I/O queues, fetch auxiliary data such as error logs, format storage devices, etc. Storage devices in NVMe are called **namespaces** and commands for a specific namespace include a namespace ID. The base specification also defines an **NVM Command Set** used for I/O requests to block-oriented namespaces. The specification is designed for future extensions including additional I/O command sets (e.g. an I/O command set targeting a key-value store). An NVMe controller and its attached namespaces together are called an **NVM subsystem**.

NVMe over Fabrics extends the original specification to enable access to NVM subsystems over a network transport instead of PCI-express similar to using iSCSI to acccess remote block storage devices as SCSI LUNs. NVMe over Fabrics supports multiple transport layers including FibreChannel, RDMA (over both iWARP and ROCE) and TCP. To handle these different transports, Fabrics includes both transport-independent extensions to the base NVMe specification as well as transport-specific bindings.

NVMe over Fabrics extends the original specification to enable access to NVM subsystems over a network transport.

Fabrics defines a new **capsule** abstraction to support NVMe commands and completions. Each capsule contains either an NVMe command or completion. In addition, a capsule may be associated with a data buffer. To support data transfers, the existing PRP entries in NVMe commands are replaced with a single NVMe SGL entry. Fabrics also replaces the shared-memory queues used for PCI-express controllers with logical completion and submission queues. Unlike PCI-express I/O queues, Fabrics queues are always explicitly paired with each submission queue tied to a dedicated completion queue. The details of how capsules and data buffers are transmitted and received on a queue pair are transport-specific, but in abstract terms, command capsules are transferred on submission queues, and completions are transferred on completion queues.

A Fabrics **host** creates an admin queue pair and one or more I/O queue pairs connected to a **controller**. A complete set of queue pairs is called an **association**. A single association may contain multiple transport-specific connections. For example, the TCP transport uses a dedicated connection for each queue pair, so an active TCP association requires at least two TCP connections.

In addition to I/O controllers which provide access to namespaces, Fabrics adds a **discovery** controller type. A discovery controller supports a new discovery log page which describes the set of controllers available in a Fabrics NVM subsystem. The log page may include one or more I/O controllers and/or references to discovery controllers in other subsystems. The log page may include multiple entries for a single controller if a controller can be accessed via multiple transports. Each log page entry contains the type of a controller (I/O or discovery) as well as the transport type and transport-specific address. For the TCP transport the address includes the IP address and TCP port number.

A Fabrics host creates an admin queue pair and one or more I/O queue pairs connected to a controller.

Fabrics hosts and controllers are identified by an NVMe Qualified Name (**NQN**). NQNs are an ASCII string which should start with "nqn.*YYYY-MM.reverse-domain*" followed by an optional trailer. The *reverse-domain* portion of the name should be a valid DNS name in reverse order, and the *YYYY* and *MM* fields should specify a valid year and month when the DNS name was owned by the organization using the prefix. The specification defines a fixed subsystem NQN for Discovery controllers as well as a scheme for constructing a NQN from a UUID. Both the host and subsystem (controller) NQNs must be specified when establishing an association.

FreeBSD 15 includes support for accessing remote namespaces via a host kernel driver as well as support for exporting local storage devices as namespaces to remote hosts. The kernel implementation of Fabrics includes a transport abstraction layer (provided by `nvmf_transport.ko`) to hide most of the transport-specific details from the host and controller modules. This module is auto-loaded as needed. Separate kernel modules provide support for individual transports. These modules must be explicitly loaded to enable use of a transport. Currently, FreeBSD includes support for the TCP transport via `nvmf_tcp.ko`. TCP specific details are documented in nvmf_tcp(4).

## Host

The Fabrics host in FreeBSD consists of new nvmecontrol(8) commands and an nvmf(4) kernel driver. The kernel driver exposes remote controllers as nvmeX new-bus devices similar to PCI-express NVMe controllers. Remote namespaces are exposed via nda(4) disk devices via CAM. Unlike the PCI-express nvme(4) driver, the Fabrics host driver does not support the nvd(4) disk driver. All of the new nvmecontrol(8) commands use a host NQN generated from the host's UUID unless an explicit host NQN is given.

## Discovery Service

The nvmecontrol(8) **`discover`** command queries the discovery log page from a discovery controller and displays its contents. Example 1 shows the log page from a Fabrics controller running on a Linux system. For the TCP transport, the service identifier field identifies the TCP port of the remote controller.

**Example 1:** The Discovery Log Page from a Linux Controller

```
# nvmecontrol discover ubuntu:4420
Discovery
=========
Entry 01
========
 Transport type:        TCP
 Address family:        AF_INET
 Subsystem type:        NVMe
 SQ flow control:       optional
 Secure Channel:        Not specified
 Port ID:               1
 Controller ID:         Dynamic
 Max Admin SQ Size:     32
 Sub NQN:               nvme-test-target
 Transport address:     10.0.0.118
 Service identifier:    4420
 Security Type:         None
```

## Connecting To an I/O Controller

The nvmecontrol(8) connect command establishes an association with a remote controller. Once the association is established, it is handed off to the nvmf(4) driver which creates a new **`nvmeX`** device. The connect command requires both the network address and subsystem NQN of the remote controller. Example 2 connects to the I/O controller listed in Example 1.

**Example 2:** Connecting to an I/O Controller

```
# kldload nvmf nvmf_tcp
# nvmecontrol connect ubuntu:4420 nvme-test-target
```

Once the association is established, the kernel outputs the text from Figure 1 to the system console and system message buffer. The **`nvmeX`** device includes the remote subsystem NQN in the device description, and each remote namespace is enumerated as an **`ndaX`** peripheral.

**Figure 1:** Console Messages from Connecting

```
nvme0: <Fabrics: nvme-test-target>
nda0:  at nvme0 bus 0 scbus0 target 0 lun 1
nda0:  <Linux 5.15.0-8 843bf4f791f9cdb03d8b>
nda0:  Serial Number 843bf4f791f9cdb03d8b
nda0:  nvme version 1.3
nda0:  1024MB (2097152 512 byte sectors)
```

The **nvme0** device from Figure 1 can be used with other nvmecontrol(8) commands such as **identify** similar to PCI-express controllers. Example 3 shows a subset of the **identify** controller data displayed by nvmecontrol(8). The **nda0** disk device can be used like any other NVMe disk device.

**Example 3:** Identify a Remote I/O Controller

```
# nvmecontrol identify nvme0
Controller Capabilities/Features
================================
...
Model Number:                Linux
Firmware Version:            5.15.0-8
...

Fabrics Attributes
==================
I/O Command Capsule Size:    16448 bytes
I/O Response Capsule Size:   16 bytes
In Capsule Data Offset:      0 bytes
Controller Model:            Dynamic
Max SGL Descriptors:         1
Disconnect of I/O Queues:    Not Supported
```

## Connecting via Discovery

The nvmecontrol(8) connect-all command fetches the discovery log page from the indicated discovery controller and creates an association for each log page entry. The association from Example 2 could be created by executing **nvmecontrol connect-all ubuntu:4420** instead of fetching the discovery log page and using the connect command.

## Disconnecting

The nvmecontrol(8) disconnect command detaches the namespaces from a remote controller and destroys the association. Example 4 disconnects the association created by Example 2. The **disconnect-all** command destroys associations with all remote controllers.

**Example 4:** Disconnecting From a Remote I/O Controller

```
# nvmecontrol disconnect nvme0
```

## Reconnecting

If a connection is interrupted (for example, one or more TCP connections die), the active association is torn down (all queues are disconnected), but the **nvmeX** device is left in a quiesced state. Any pending I/O requests for remote namespaces are left pending as well. In this state, the reconnect command can be used to establish a new association to resume operation with a remote controller. Example 5 reconnects to the controller from Example 2. Note that the reconnect command requires an explicit network address similar to the connect command.

**Example 5:** Reconnecting to a Remote I/O Controller

```
# nvmecontrol reconnect nvme0 ubuntu:4420
```

## Controller

The Fabrics controller on FreeBSD exposes local block devices as NVMe namespaces to remote hosts. The controller support on FreeBSD includes a userland implementation of a discovery controller as well as an in-kernel I/O controller. Similar to the existing iSCSI target in FreeBSD, the in-kernel I/O controller uses CAM's target layer (ctl(4)).

Block devices are created by adding ctl(4) LUNs via ctladm(8). The discovery service and initial handling of I/O controller connections are managed by the nvmfd(8) daemon. The in-kernel I/O controller is provided by the nvmft(4) module. Example 6 adds a ZFS volume named pool/lun∅ as a ctl(4) LUN and starts the nvmfd(8) daemon. Remote hosts can then access the ZFS volume as a NVMe namespace.

**Example 6:** Exporting a local ZFS Volume

```
# kldload nvmft nvmf_tcp
# ctladm create -b block -o file=/dev/zvol/pool/lun0
LUN created successfully
backend:       block
device type:   0
LUN size:      4294967296 bytes
blocksize      512 bytes
LUN ID:        0
Serial Number: MYSERIAL0000
Device ID:     MYDEVID0000
# nvmfd -F -p 4420 -n nqn.2001-03.com.chelsio:frodo0 -K
```

Each time a remote host connects to the I/O controller, a message is logged by the kernel listing the remote host's NQN (see Figure 2).

**Figure 2: Log Messages from New Association**

```
nvmft0: associated with
nqn.2014-08.org.nvmexpress:uuid:00000000-0000-0000-0000-ffffffffffff
```

LUNs can be added or removed by ctladm(8) while nvmfd(8) is running. If any remote hosts are connected while a LUN is added or removed, an asynchronous event is reported to the remote hosts. This allows remote hosts to notice that namespaces have been added or removed while connected.

Two new commands have been added to ctladm(8) to manage Fabrics associations. The **nvlist** command lists all active associations from remote hosts. Example 7 shows the output from the **nvlist** command while a single host is connected to the controller from Example 6.

**Example 7:** Listing Active Associations

```
# ctladm nvlist
  ID Transport       HostNQN                                     SubNQN
   0 TCP
nqn.2014-08.org.nvmexpress:uuid:00000000-0000-0000-0000-ffffffffffff
nqn.2001-03.com.chelsio:frodo0
```

The `nvterminate` command closes one or more associations. Associations for a single connection or NQN can be terminated, or all active associations can be terminated. Example 8 terminates the association from Example 7. After the association is terminated, the kernel logs the messages from Figure 3.

**Example 8:** Terminating an Association

```
# ctladm nvterminate -c 0
NVMeoF connections terminated
```

**Figure 3: Log Message after Terminating**

```
nvmft0: disconnecting due to administrative request
nvmft0: association terminated
```

## Conclusion

NVMe over Fabrics support will be available in FreeBSD 15.0 including both host and controller support. The development of Fabrics support was sponsored by Chelsio Communications, Inc.

**JOHN BALDWIN** is a systems software developer. He has directly committed changes to the FreeBSD operating system for over twenty years across various parts of the kernel (including x86 platform support, SMP, various device drivers, and the virtual memory subsystem) and userspace programs. In addition to writing code, John has served on the FreeBSD core and release engineering teams. He has also contributed to the GDB debugger. John lives in Concord, California with his wife, Kimberly, and three children: Janelle, Evan, and Bella.