Implementing NVMe over Fabrics in FreeBSD

John Baldwin

EuroBSDCon

16 September 2023

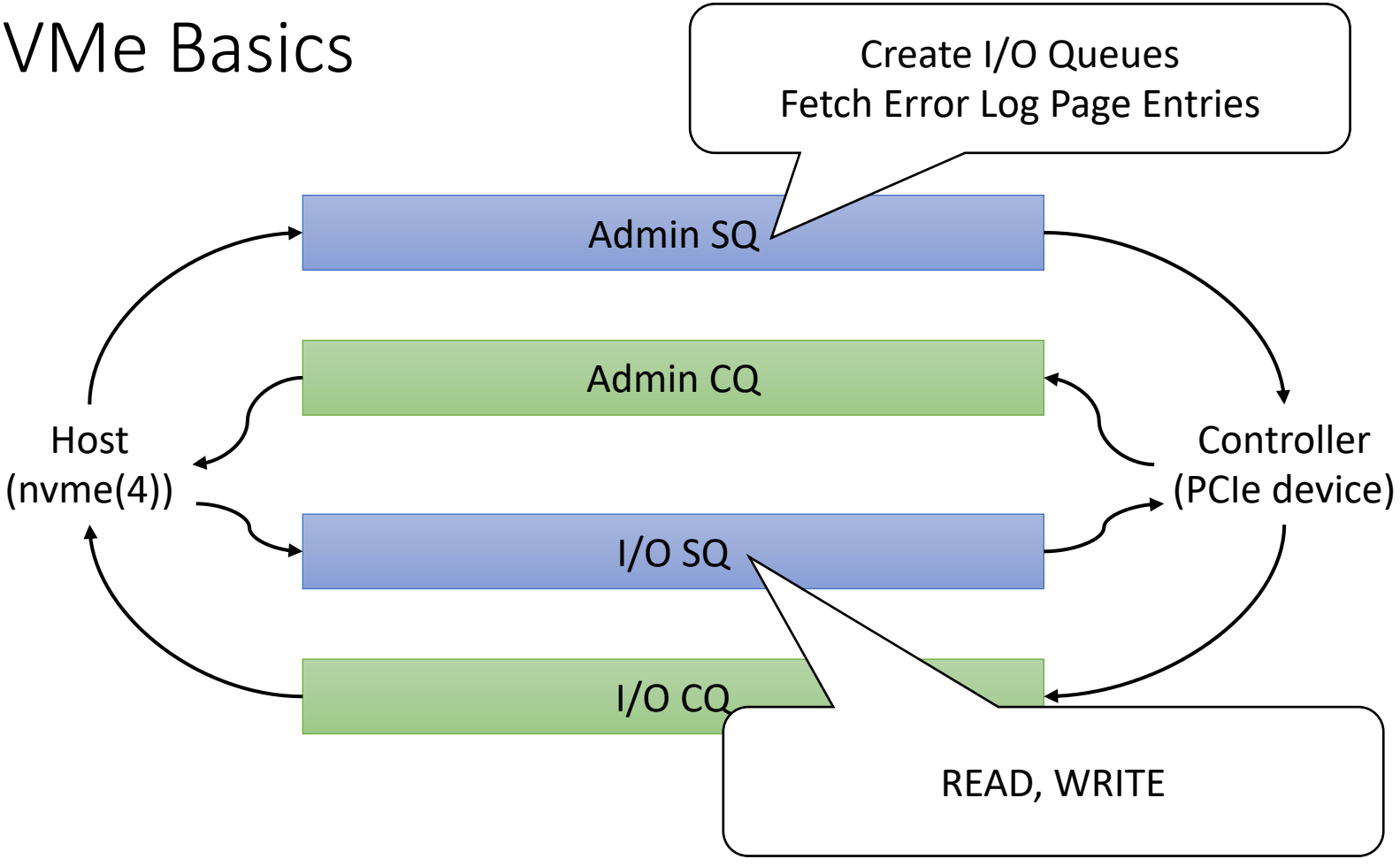
Overview

- Introduction to NVMe and NVMe over Fabrics
- FreeBSD Implementation
  - Three Layer Design
  - Userspace Library and Tools
  - Kernel Datapath
- Future Work
- Demo

NVMe Basics

- Storage devices which use a command protocol somewhat similar to SCSI and ATA
- Host (e.g. OS driver) sends commands to a controller in FIFO submission queues (SQs)
- Controller sends completions back to the host on completion queues (CQs)
- Queue entries are fixed-size
  - Submission Queue Entries (SQEs): 64-byte commands
  - Completion Queue Entries (CQEs): 16-byte responses
- Admin queues handle administrative commands
- I/O queues handle I/O commands like READ and WRITE

NVMe Basics


Create I/O Queues
Fetch Error Log Page Entries
Admin SQ
Admin CQ
Host
(nvme(4))
Controller
(PCIe device)
I/O SQ
I/O CQ
READ, WRITE

NVMe Commands

- Commands are a fixed size (64-byte SQE)
- Commands do not embed I/O data, but instead store a scatter/gather list
- NVMe over PCIe uses a specialized S/G list where each element is just an address of a page called a Physical Region Page (PRP)
- Commands embed two adjacent PRP entries
- NVMe also defines a more traditional S/G list type (SGL) where each element includes both an address and length as well as a type
  - Not typically used for PCI-express controllers

NVMe Completions

- Completions are a fixed size (16-byte CQE)
- Completions do not embed I/O data
- If a request needs to return data to the host, the associated command must provide data buffer in SQE and controller stores data before sending the CQE
- Completions are matched to submitted commands via Command IDs

NVMe over Fabrics

- Replaces SQs and CQs stored in memory with queues implemented on top of a transport
- Currently defined for several transports: Fibre Channel, RDMA, and TCP
- Each SQ always associated with a dedicated CQ
  - Referred to as a SQ/CQ pair or queue pair in the rest of this talk
- A logical connection between a host and controller (admin queue pair and one or more I/O queue pairs) is called an “association”
- Adds a discovery controller type with a new discovery log page

NVMe over Fabrics Capsules

- Commands (SQEs) and Completions (CQEs) are embedded in capsules
  - Command Capsules for SQEs
  - Response Capsules for CQEs
- Capsules may be associated with a data buffer
- Fabrics commands always use SGL to describe data buffer, never PRP
- Data may be embedded in the capsule (In-Capsule Data or ICD) following the Command (SQE) or Completion (CQE)
- Data may be stored in a logical buffer managed by the transport
  - Data must be read/written from buffer by controller before sending response capsule

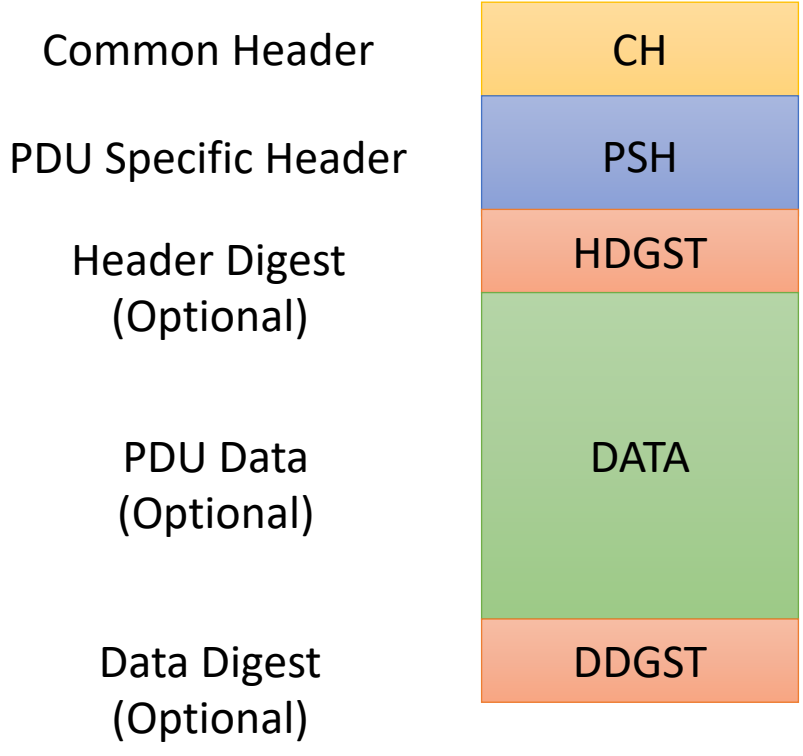
NVMe over TCP

- Uses a lower-level message protocol that passes Protocol Data Units (PDUs)
- Very similar to iSCSI's TCP transport
- Uses a separate TCP connection for each SQ/CQ pair
  - Completions received on the same TCP connection that sent the command
- Supports In-Capsule Data (ICD) for Command Capsules only
- Supports a Command Buffer abstraction for data buffers associated with SGL in a Command

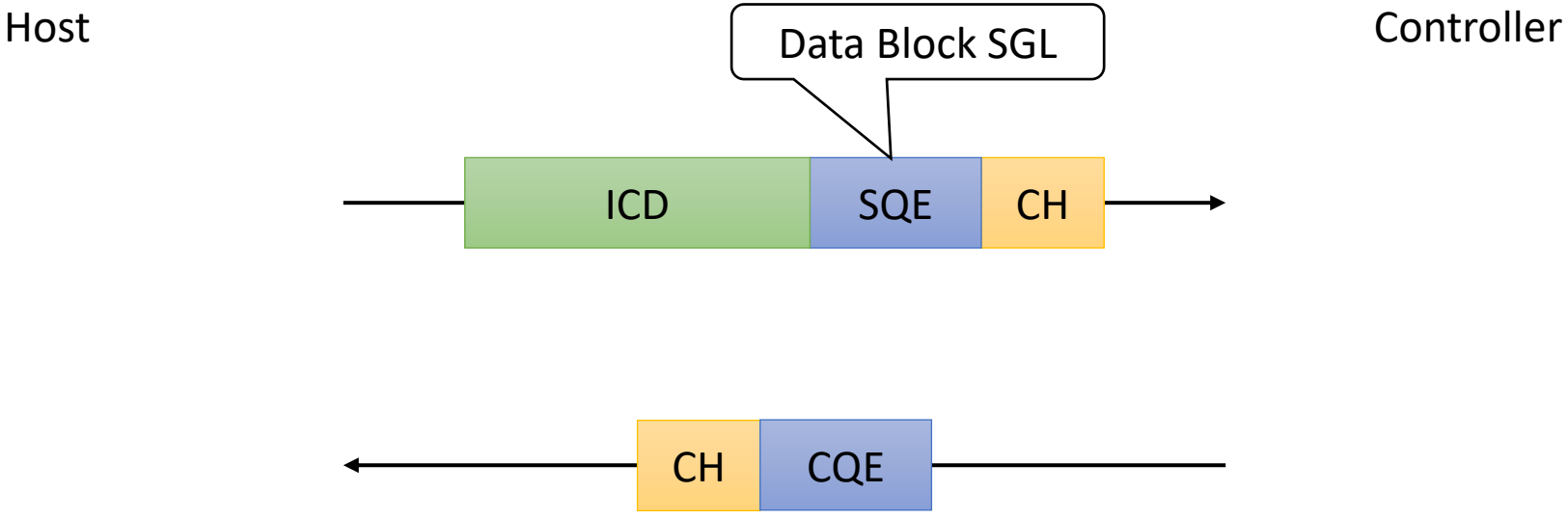
NVMe/TCP PDU Types

Name	Description
ICReq and ICResp	Connection establishment (negotiate digests, etc.)
H2CTermReq and C2HTermReq	Terminate connection due to NVMe/TCP protocol error
CapsuleCmd	Command capsule with SQE and optional ICD
CapsuleResp	Response capsule with CQE
H2CData	Host to Controller data
C2HData	Controller to Host data
R2T	Controller is ready for Host to transmit data

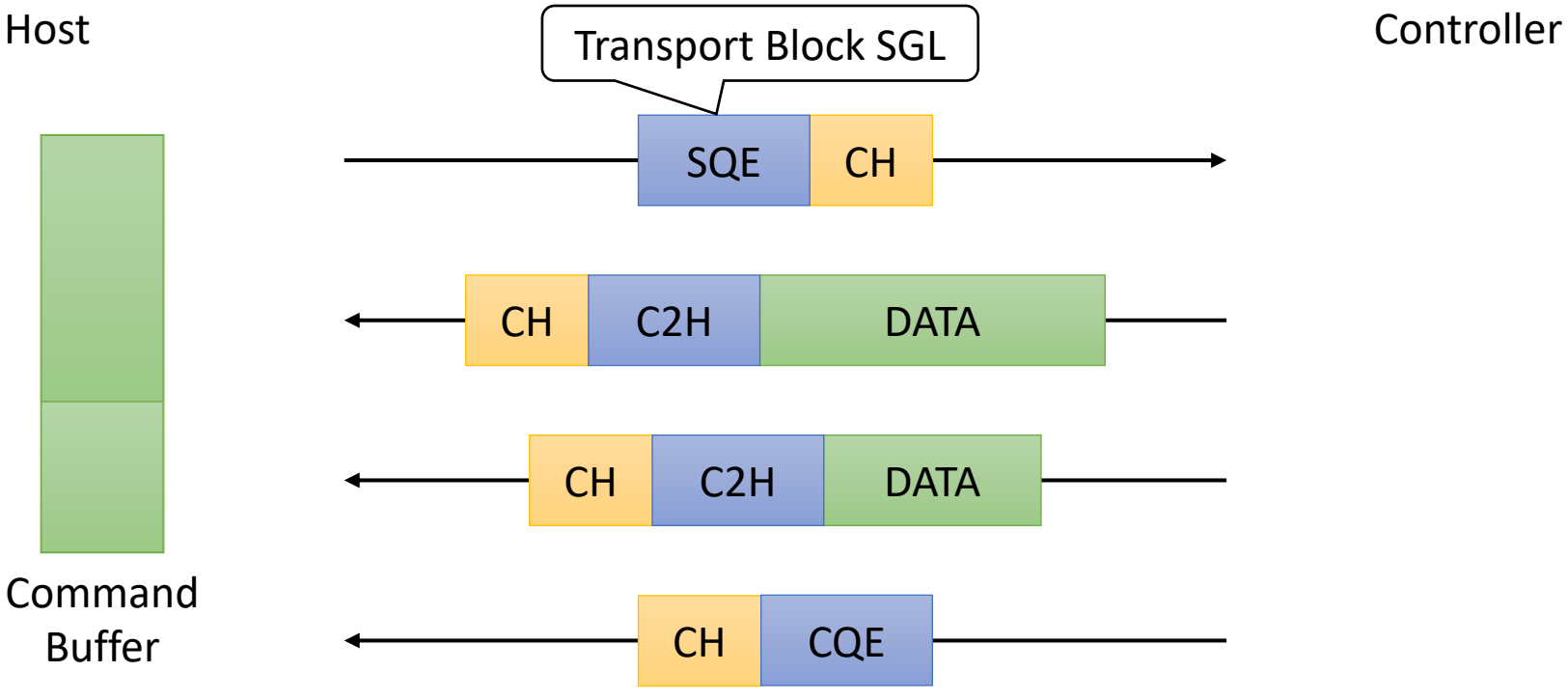
NVME/TCP PDUs



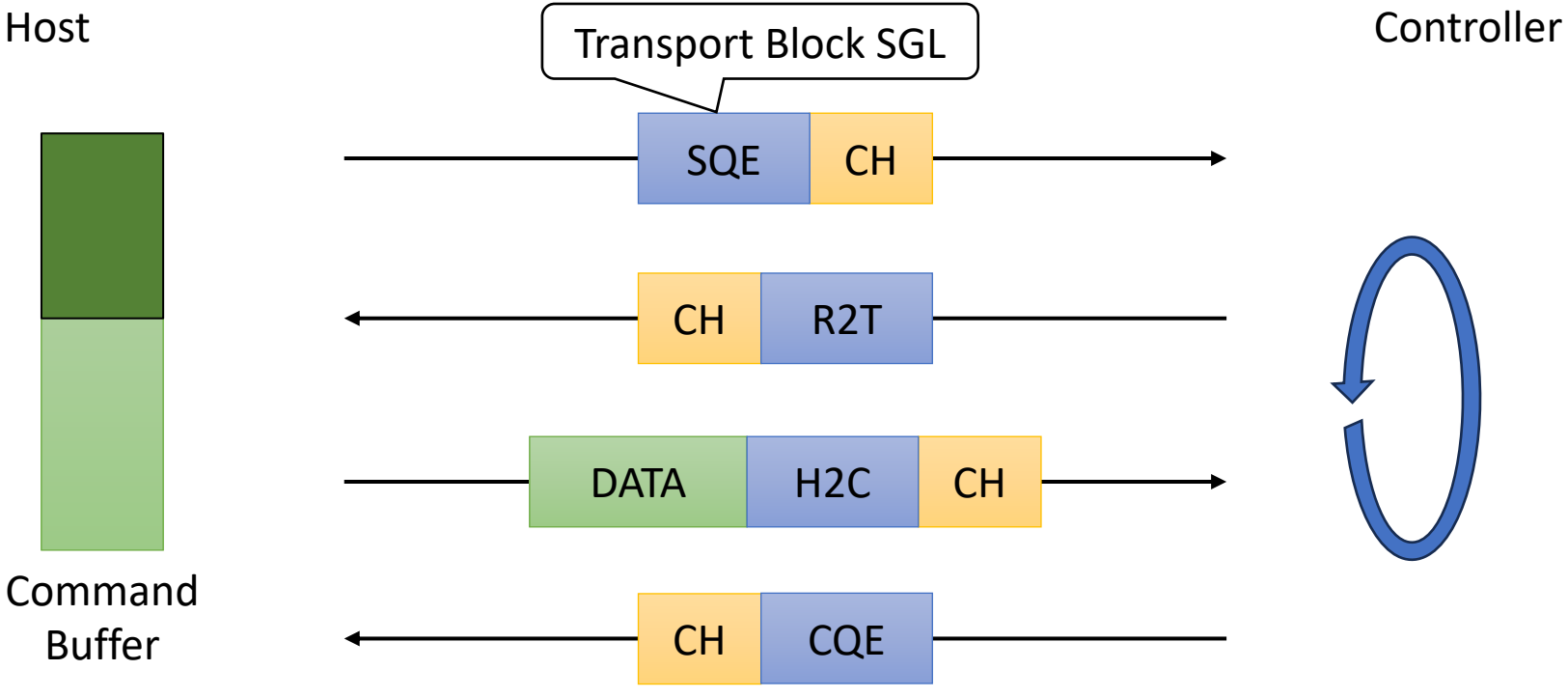
NVMe/TCP: Sending data via ICD



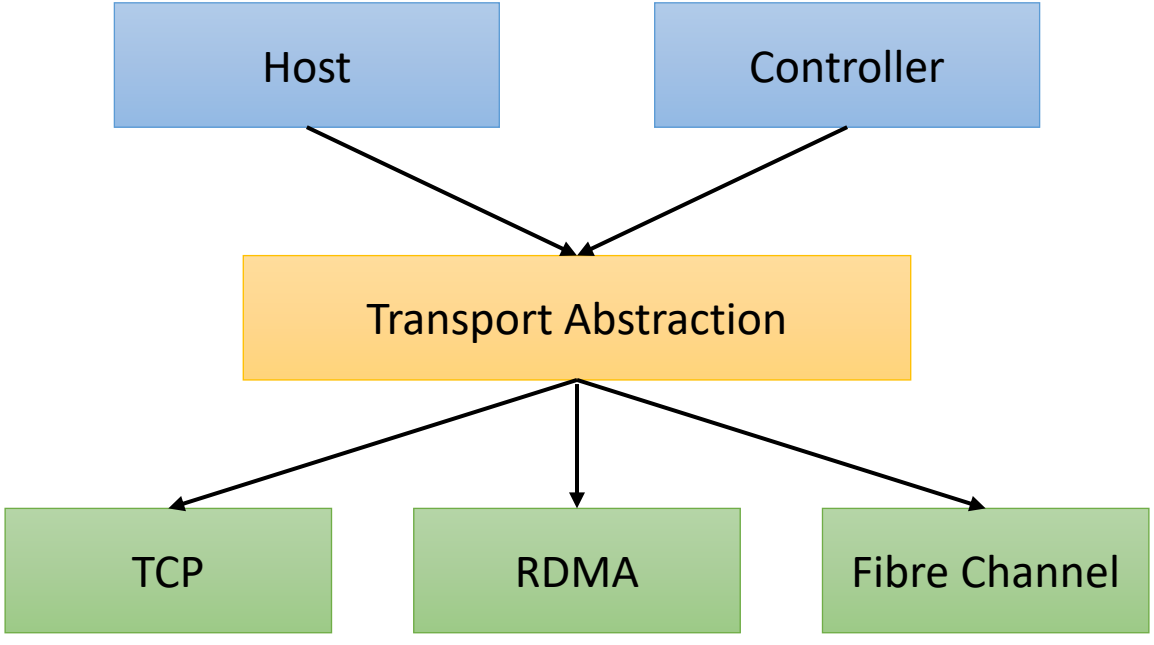
NVMe/TCP: Receiving data via Command Buffer



NVMe/TCP: Sending data via Command Buffer



FreeBSD Implementation: Three Layer Design



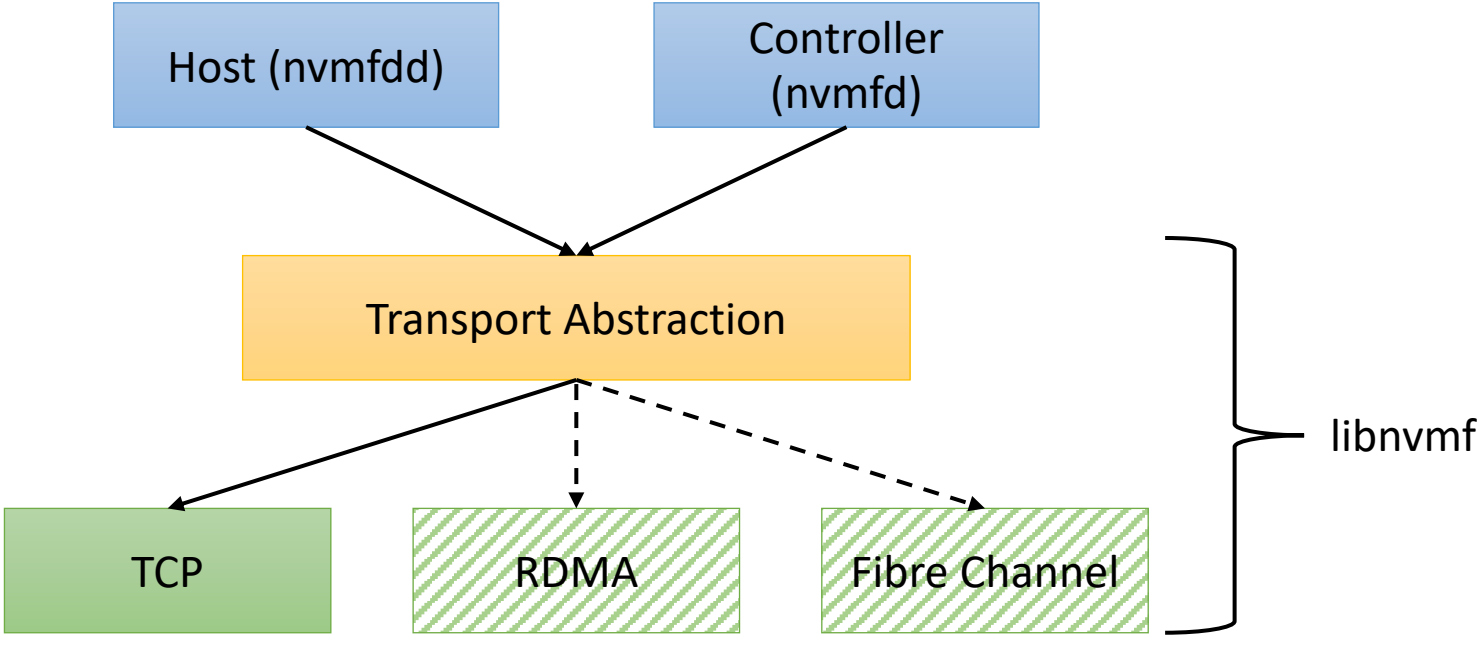
Userspace library: libnvmf

- Defines a transport abstraction interface to send and receive capsules
- Provides an implementation of the TCP transport
- Designed for simplicity, not necessarily performance
  - Not thread-safe
  - Uses blocking I/O on sockets
- Contains some helper routines on top of the transport abstraction both for hosts and controllers

Userspace Host and Controller

- nvmfdd is a simple userspace host that can read or write from a single namespace on a remote controller providing similar function to dd(8)
- nvmfd is a simple userspace controller
  - Supports multiple namespaces backed either by a file, character device, or memory buffer
  - Implements a discovery controller as well as an I/O controller
- Not designed for performance, but much easier to debug and uncover incorrect assumptions testing these first before moving into the kernel

Three Layers in Userspace



Kernel Datapath

- Mirrored the transport abstraction from libnvmf into the kernel
  - Some regrettable code duplication
- Uses asynchronous callbacks instead of blocking
  - Callback when a capsule is received
  - Callback when an I/O operation (e.g. reading or writing to capsule data buffer) completes
  - Callback if an error occurs on a queue pair
- I/O buffers attached to capsules represented by struct memdesc
- Userspace should still perform initial setup of queue pairs and then hand them off to the kernel

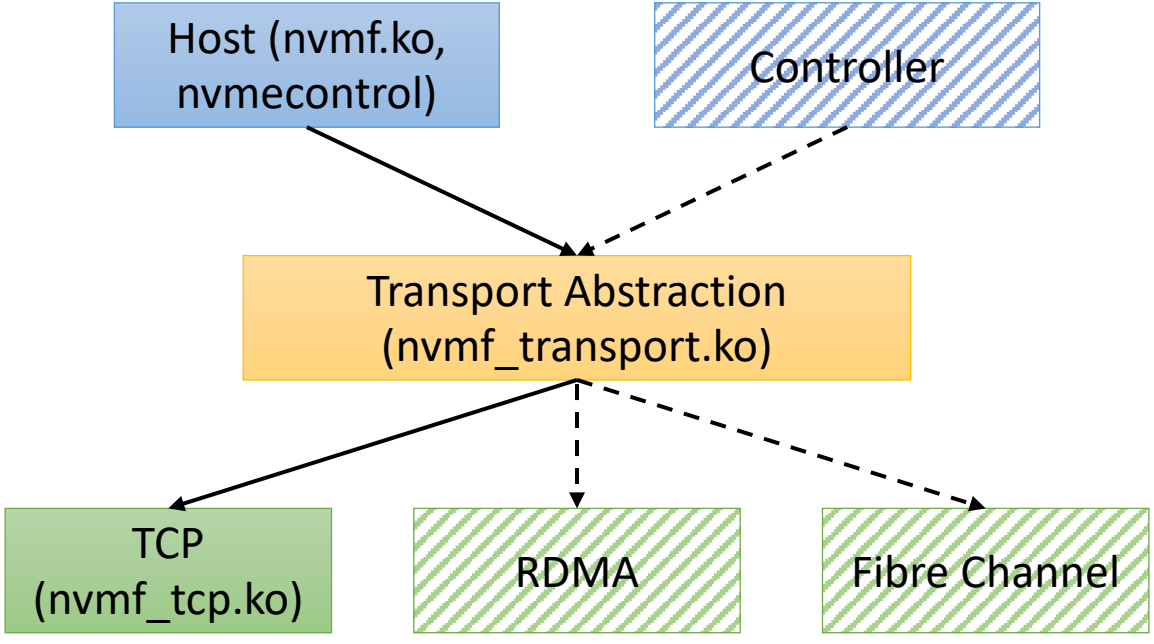
Host: nvmf(4)

- nvmf(4) provides an in-kernel Fabrics host
- Does not try to share code with nvme(4)
- Creates nvmeX new-bus devices for each host
- Creates /dev/nvmeX and /dev/nvmeXnsY device nodes like nvme(4)
  - nvmecontrol(8) works including passthrough commands
- Only supports disk access via CAM (ndaX disks)
- If a connection error occurs, existing I/O operations are paused and the queue pairs are destroyed
  - I/O is resumed if a new association is established with the same controller

Host: nvmecontrol(8) extensions

- Identify controller command now displays Fabrics-specific fields
- New “discover” command connects to a remote discovery controller and displays the Discovery Log Page listing remote controllers
- New “connect” command connects to a remote I/O controller creating admin and I/O queue pairs and handing them off to nvmf(4) to create a new nvmeX device
- New “disconnect” command detaches an nvmeX device closing its associated queue pairs
- New “reconnect” command connects to a remote I/O controller creating admin and I/O queue pairs to restore a nvmeX device

Three Layers in the Kernel



Future Work

- In-kernel Controller
  - Would use ctl(4) LUNs as backing store for namespaces
  - Initially configured by ctladm(8)
  - Will require extending ctl(4) to support NVMe I/O CCBs
  - Discovery controller support in nvmfd will remain in userland, I/O controller support would move into the kernel
- Other transports such as RDMA or Fibre Channel
- TLS protection for TCP queue pairs
  - Requires KTLS for kernel datapath

Demo

- Testing nvmf(4) host against a remote target on a Linux VM running Ubuntu 22.04 (“ubuntu”)

Demo: nvmecontrol discover

```
# 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
```

Demo: nvmecontrol connect

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

…
<dmesg>
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)
```

Demo: nvmecontrol identify (1)

```
# 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
```

Demo: nvmecontrol identify (2)

```
# nvmecontrol identify nvme0ns1
Size:                           2097152 blocks
Capacity:                       2097152 blocks
Utilization:                    2097152 blocks
Thin Provisioning:              Not Supported
Number of LBA Formats:          1
Current LBA Format:             LBA Format #00
…
LBA Format #00: Data Size:   512
  Metadata Size:     0  Performance: Best
```

Demo: Connection Error

```
# tcpdrop -la | grep 118 | head -1 | sh
10.0.0.121 57894 10.0.0.118 4420: dropped

…
<dmesg>
nvme0: error on I/O queue 0, disconnecting
nvme0: error on I/O queue 0, disconnecting
```

Demo: nvmecontrol reconnect

```
# nvmecontrol reconnect nvme0 ubuntu:4420 nvme-test-target


…
<dmesg>
nvme0: established new association with 1 I/O queues
```

Demo: nvmecontrol disconnect

```
# nvmecontrol disconnect nvme0


…
<dmesg>
nda0 at nvme0 bus 0 scbus0 target 0 lun 1
nda0: <Linux 5.15.0-8 843bf4f791f9cdb03d8b> s/n
843bf4f791f9cdb03d8b detached
(nda0:nvme0:0:0:1): Periph destroyed
nvme0: detached
```

Conclusion

- Code is available in the ”nvmf2” branch at https://github.com/bsdjhb/freebsd.git
  - Caveat: I will probably rebase often until it is merged into “main”
- Thanks to Chelsio Communications for sponsoring this work
- Questions?

libnvmf Data Structures

- struct nvmf_association_params: Parameters shared a group of queue pairs
  - Includes transport protocol (e.g. TCP)
  - Includes transport-specific params (e.g. whether to use digests for TCP)
- struct nvmf_association: Represents a group of related queue pairs
  - For a host, all of the queues for a single association share a single instance
  - For a controller, all queues of the same controller type share a single instance

libnvmf Data Structures

- struct nvmf_qpair_params: Parameters specific to a single SQ/CQ pair
  - Admin vs I/O
  - For TCP, contains file descriptor for socket
- struct nvmf_qpair: Represents a SQ/CQ pair
  - For a host, nvmf_connect() allocates a queue pair and connects to the controller via Fabrics CONNECT command
  - For a controller, nvmf_accept() allocates a queue pair and waits for the CONNECT command from the remote host

libnvmf Data Structures

- struct nvmf_capsule: Represents either a Command or Response capsule
  - nvmf_allocate_command() allocates a Command capsule containing the supplied SQE
  - nvmf_allocate_response() allocates a Response capsule containing the supplied CQE
- A data buffer can be attached to a Command capsule via nvmf_capsule_append_data()
  - This data buffer is used to transfer data in a transport-specific manner
  - For TCP the buffer contents can be sent as ICD or used as a Command Buffer