OBJECTIVE (TODAY)

- Describe what we’re trying to accomplish, and its rationale

- Describe the approach being taken

- Ask for your feedback/direction check - Is this an acceptable direction that merits further development?
Pathfinding a Kernel Storage Fabric Mid-layer

Scott Atchley, Stan Smith, Paul Grun
Kernel Fabric Observations

- Kernel fabric clients (ULPs) require a fabric device specific bottom edge in order to interface with kernel fabric devices.

- Each ULP is forced to define a fabric transport abstraction layer and then meld the fabric device specific behavior into their fabric transport abstraction; often 8+ months of development work.

Case in point:
- LNET is the fabric transport abstraction
- A Lustre Network Driver (LND) is required for each supported fabric: LNDs for IB/iWARP, Cray GNI, ksockets, non-QP based devices...

- NVMe over Fabrics (NVMe/F), NFS/RDMA, iSER/SRP have the same ULP-to-fabric device I/F issues as Lustre in order to support new interconnects.
Current Lustre LND Architecture

- **Vendor Driver**
- **Supported by Lustre Community**
- **Supported by HW Vendors**
- **Supported by OpenFabrics Community**
Pathfinding Conclusions

- Reduction in storage ULP fabric device I/F development time for new fabric devices is desirable

- Multiple storage ULPs could utilize a common fabric mid-layer

- A storage fabric mid-layer would be RMA device agnostic in order to support current and future RMA devices
  - Not all fabric devices are Queue-Pair based
  - Support diverse fabrics w/o requiring emulation of an existing fabric (i.e. not wire compatible)

- Fabric mid-layer would present a consumer-oriented message transfer abstraction
  - Minimize device specific special cases above message transfer layer

- Support emerging fabric use cases – NVMe for remote storage (NVMe/F)
  - NVMe is PCIe slot and device [0…255] limited
  - NVMe/F gains access to ‘more’ NVMe resources at ‘near local’ speeds
  - Sharing NVMe data over the fabric
  - Data replication / Mirroring using RMA (multicast+) especially for NVDIMMs
  - All RMA writes must reach a durability point before signaling completion
Hold on, it’s not a QP device...
Fabric Mid-layer Objectives

- Kernel storage ULP I/F requirements drive fabric mid-layer messaging API design
  - File systems, object I/O, block storage, persistent memory (emerging)

- Fabric agnostic
  - Support for new fabrics should not require emulating an existing one
    - Device drivers are typically based on a specific fabric technology

- Support for emerging fabrics…
  - Allow for innovation from new fabrics as they emerge

- While still supporting existing networks
  - Must be able to support existing network technologies
kfabric Mid-layer Proposal

- **kfabric**: an abstract, kernel mode API for storage
  - API is expressed in terms of message passing operations, not fabric device protocols (e.g. ‘write message’ vs ‘post send request’)
  - Fabric provider does address resolution in consumer-provider agreed upon address format

- Emerging NVMe/F technology can benefit from a transport neutral, RMA-enabled fabric mid-layer

- **kfabric designed in ‘spirit’ around libfabric concepts**
  - RMA device agnostic (consider SCSI mid-layer common code design)
  - Reduce/Simplify ULP fabric device specific I/F code
    - Device specifics contained in the provider module, not in ULP
    - NVMe/F and Lustre LND fabric I/F implementations reap benefits (code reduction/simplification) from kfabric mid-layer

Demand exists for an abstract, fabric mid-layer API based on RMA
kfabric Mid-layer Framework

- kfabric API
- kfabric Providers
- Device Drivers

New Providers**

New Devices

Sockets Provider
Kernel Sockets

Verbs Provider
Kernel Verbs

NIC
InfiniBand
iWarp
RoCE

Red = new kernel components, ** = e.g. NVM
**kfabric consumer API module (exports)**

- `fi_getinfo()`
- `fi_fabric()`
- `fi_domain()`
- `fi_endpoint()`
- `fi_cq_open()`
- `fi_ep_bind()`
- `fi_listen()`
- `fi_accept()`
- `fi_connect()`
- `fi_send()`
- `fi_recv()`
- `fi_read()`
- `fi_write()`
- `fi_mr_reg/v()`
- `fi_cq_read()`
- `fi_cq_sread()`
- `fi_eq_read()`
- `fi_eq_sread()`
- `fi_close()` …

**kfabric Provider APIs**

- Each fabric device type is implemented as a kfabric device provider module.

- **`kfi_provider_register()`**
  During kfabric provider module load, a call to `kfi_provider_register()` supplies the kkgabric API with dispatch vectors for `fi_*` calls to the provider specific routines.

- **`kfi_provider_deregister()`**
  During kfabric provider module unload, `kfi_provider_deregister()` destroys the `fi_*` runtime linkage for the specific provider (ref counted).
Why a Kernel Storage Fabric Mid-Layer

- **Reliable sockets is a byte streaming interface**
  - Semantics do not map well to messaging operations (i.e. msg markers required)
    - kfabric complements sockets by providing a reliable message service
  - And sockets does not scale well in time or space
    - Polling connections for progress or memory consumption per connection

- **Kernel verbs is a low-level device driver I/F**
  - Not just an complicated interface, but also wire protocols (IB, RoCE, iWarp)
  - Lacking stronger completion semantics (i.e. data resides within a persistence domain)
  - kfabric is expected to call kverbs for certain networks

- **An RMA device agnostic fabric mid-layer does not exist today**

The semantics desired by current and emerging storage applications are not completely addressed by current APIs
Block and object storage protocols map well to reliable message-based APIs that provide RMA services

- kfabric provides reliable and unreliable message services
  - Fabric clients do not need to maintain message markers

- kfabric does not require implicit buffering

- kfabric completion semantics are a semantic match with storage requirements
  - e.g. Completions: local, remote, persistent, ordered and out-of-order data delivery…

- kfabric endpoints are thread-safe (when requested)
  - Multiple threads can make forward progress independently
  - Serialization can be done by the provider, not by the application/ULP

- kfabric provides one-sided semantics enabling hardware Remote Memory Access without remote CPU intervention
Current Lustre LND Architecture

- o2ibInd
- OFED
- LNet
- sockInd
- Ethernet
- gnilnd

Vendor Driver:

- Supported by Lustre Community
- Supported by HW Vendors
- Supported by OpenFabrics Community
Future Lustre LND Architecture

- LNet
- kFabric
- Provider
- verbs Provider
- gni Provider
- sock Provider
- Vendor Driver
- Vendor Driver

**Supported by**
- LNet: Supported by Lustre Community
- kFabric: Supported by HW Vendors
- Provider: Supported by OpenFabrics Community
THANK YOU

OFIWG – DS/DA