# Basic Proposal

Provide mechanisms to allow a programming environment (MPI, SHMEM…) to override libfabric access methods for accessing device memory.

The existing mechanisms work, and are acceptable. But in some cases there may be more efficient mechanisms for doing so.

# Motivation

* Design/Implementation choices:
	+ Providers and/or utility layers offer basic GPU functionality
	+ Specific providers can handle additional GPU/NIC integration nuances
	+ Interfaces to allow clients to implement certain overrides
* Motivation for override interfaces:
	+ Support a broad range of GPUs and accelerators technologies:
		- Clients can use call-backs/overrides to support upcoming technologies
	+ Use cases where clients may benefit from overrides:
		- Datatype optimizations (in addition to the current iovec infrastructure)
		- Certain collective operations (fi\_collective support is still experimental)
		- Tuning and other optimizations for proprietary implementations of client programming models
* Collectives
	+ MPI libraries have invested a lot of effort in developing a range of algorithms for each operation.
	+ Can we use the callback/override idea to point to ways these existing algorithms/implementations can be leveraged instead of reinventing them all inside the provider?

# Detailed Proposal

## Overview

The preferred override approach is to define a libfabric override structure which the libfabric user can optionally use to override internal libfabric provider calls. The override structure is associated with the libfabric domain. If a call in override structure is defined, the domain and child objects (MRs, EPs, etc...) will all use the override calls. If a call is not defined, domain and child objects will use the libfabric provider default implementation.

Note: For this override approach to work, libfabric providers must honor the user defined callbacks.

The following sections go into detail about the proposed implementation.

## 1. Libfabric Override Structure

This libfabric override approach defines a new override structure consisting of pointers defining function signatures. The following is an example of this override structure consisting of two HMEM IOV override options.

|  |
| --- |
| struct fi\_override\_ops {    ssize\_t (\*copy\_from\_hmem\_iov)(**void** \*dest, size\_t size, **const** struct iovec \*hmem\_iov, **enum** fi\_hmem\_iface \*hmem\_iface, size\_t hmem\_iov\_count, uint64\_t hmem\_iov\_offset);    ssize\_t (\*copy\_to\_hmem\_iov)(**const** struct iovec \*hmem\_iov, **enum** fi\_hmem\_iface \*hmem\_iface, size\_t hmem\_iov\_count, uint64\_t hmem\_iov\_offset, **void** \*src, size\_t size);}; |

The following is a list of rules the provider must adhere to when processing the override arguments:

1. If a function pointer is NULL, the provider must use the default libfabric/provider implementation.
2. If a function pointer is non NULL, the provider must use the provider function pointer instead of the default libfabric/provider implementation.

**Note:** This behaviour creates the optional, user-defined override behaviour.

## 1.1. copy\_from\_hmem\_iov Override

The copy\_from\_hmem\_iov override allows libfabric users to define how libfabric/providers copy an HMEM (e.g. host, device, etc..) IOV into a host flat buffer. For each HMEM IOV, a corresponding HMEM interface type is provided. Today, two HMEM interfaces are defined (FI\_HMEM\_SYSTEM and FI\_HMEM\_CUDA). On return, the number of bytes copied into the host flat buffer is returned or an libfabric errno value.

Note: It is acceptable to mix HMEM interface types within a single copy\_from\_hmem\_iov call.

Note: When overriding this call, it is expected that the libfabric user internally defines operations for all libfabric defined HMEM interface types. Meaning, if a new HMEM interface type is defined, the libfabric user needs to support this new type. But, if a libfabric user's override does not support the new HMEM interface type, it is acceptable to return a libfabric errno value (e.g. -FI\_ENOSYS) to the user. This will most like result in a failed libfabric data transfer.

Note: This is a synchronous call.

### 1.2. copy\_to\_hmem\_iov Override

The copy\_to\_hmem\_iov override allows libfabric users to define how libfabric/providers copy a host flat buffer into an HMEM (e.g. host, device, etc..) IOV. For each HMEM IOV, a corresponding HMEM interface type is provided. Today, two HMEM interfaces ares defined (FI\_HMEM\_SYSTEM and FI\_HMEM\_CUDA). On return, the number of bytes copied into the HMEM IOV is returned or an libfabric errno value.

Note: It is acceptable to mix HMEM interface types within a single copy\_to\_hmem\_iov call.

Note: When overriding this call, it is expected that the libfabric user internally defines operations for all libfabric defined HMEM interface types. Meaning, if a new HMEM interface type is defined, the libfabric user needs to support this new type. But, if a libfabric user's override does not support the new HMEM interface type, it is acceptable to return a libfabric errno value (e.g. -FI\_ENOSYS) to the user. This will most like result in a failed libfabric data transfer.

Note: This is a synchronous call.

## 2. Setting Overrides with a Libfabric Domain

The proposed way a libfabric user associates the override structure with a libfabric domain is through a new domain function. The following is an example of this interface.

|  |
| --- |
| **int** fi\_domain\_set\_overrides\_ops(struct fid\_domain \*domain, struct fi\_override\_ops \*ops); |

If a provider does not support the overriding of operations, -FI\_ENOSYS will be returned.

The other approach considered was to set the overrides during domain allocation. But, this would either involve changing or defining a new domain allocation interface or associating the override operations with the FI info structure. Neither seemed like good options.

## 3. Expanding Libfabric Overrides

Currently, only two overrides have been requested: copy to/from an HMEM IOV. In the future as new overrides are defined, the override structure can easily be updated to define new overrides. When this occurs, libfabric providers must be updated to honor the new override (assuming that they will be using the new override).