Libfabric (OFI)

Table of Contents

The UET reference implementation will be provided as a Libfabric provider [1]. Therefore, understanding Libfabric and OFI is required to fully understand the UET technology stack.

It is not necessary to understand all of Libfabric to understand UEC. If you are interested in the overall picture of UEC technology and its specific technologies, you can get a rough understanding of the Libfabric architecture and then read the details later as needed.

References

Libfabric and OFI are widely used in the field of HPC, and their official documentation and conference presentation materials are also publicly available. Therefore, if you ask an AI such as ChatGPT, it will likely introduce you to many reference materials, but for reference, I will list some of the materials I referred to when writing this article.

- "Libfabric Programmer's Manual" [2]
- fi_guide(7) Libfabric Programmer's Guide
  - https://ofiwg.github.io/libfabric/main/man/fi_guide.7.html
  - Introduction `fi_intro(7)` ... A detailed explanation of the background to the development of Libfabric
  - Architecture `fi_arch(7)` ...Libfabric architecture explanation (including the role of each component)
- Tutorial: OFI Libfabric, HOTI - Hot Interconnects Symposium
  - HOTI 2024: https://www.youtube.com/watch?v=Sde8Vc8dZcM
  - HOTI 2023: https://www.youtube.com/watch?v=b-ri1mMLbZ0
  - HOTI 2022: https://www.youtube.com/watch?v=ajTWxdMTvKM
- 2024 OFA Virtual Workshop Sessions
  - https://www.openfabrics.org/2024-ofa-virtual-workshop-agenda/
  - Session 1 "OFI 2.0 Update"
  - Session 13 "Designing In-Network Computing Aware Reduction Collectives in MPI"
  - Session 10 "System Composability Using CXL"

Relationship between Libfabric, OFI, and OFA

In a nutshell, OFI is the API specification, Libfabric is the API implementation, and OFA is the name of the organization that develops and promotes them. (OFI and Libfabric API are sometimes used interchangeably.)

OFA (OpenFabrics Alliance)

- Founded in 2004 under the name "OpenIB Alliance," the goal was to develop a vendor-neutral InfiniBand software stack that runs on Linux. It subsequently expanded support for Windows and protocols, adding iWarp in 2006 and RoCE in 2010. In 2014, it established the OpenFabrics Interfaces working group to support other high-performance network technologies.
- Our current vision is to provide a cross-platform, transport-agnostic software stack for RDMA and kernel bypass, where "transport-agnostic" means providing APIs that allow applications to run over InfiniBand, iWARP, RoCE or other fabric protocols transparently.

OFI (Open Fabric Interface)

- OFI is an API specification designed for efficient, high-performance network communication, providing an abstraction layer that allows applications to communicate without depending on the details of the network (differences in hardware and protocols).
- Specifically, it provides APIs for sending and receiving tagged messages (Tagged Messages), remote memory access (RMA), atomic operations (Atomics), collective communication (Collectives), and more.
- OFI has two versions, 1.x and 2.0, and UET is designed based on OFI 2.0.

Libfabric

- Libfabric is a library that provides an implementation of OFI and is developed based on the OFI API specification.
- Libfabric provides a common API for transport technologies and is an architecture that can support a variety of transport technologies by adding plugins called "libfabric providers".
- Examples of transport technologies include TCP, UDP, RDMA Verbs, EFA, SHM, and UET.

Libfabric Architecture

This article explains the architecture of Libfabric, an implementation of OFI, the various services it provides to applications, and other related components.

The Libfabric architecture diagram was created by the author with reference to the Libfabric Programmer's Manual [3] and the NANOG 92 Keynote presentation [4].

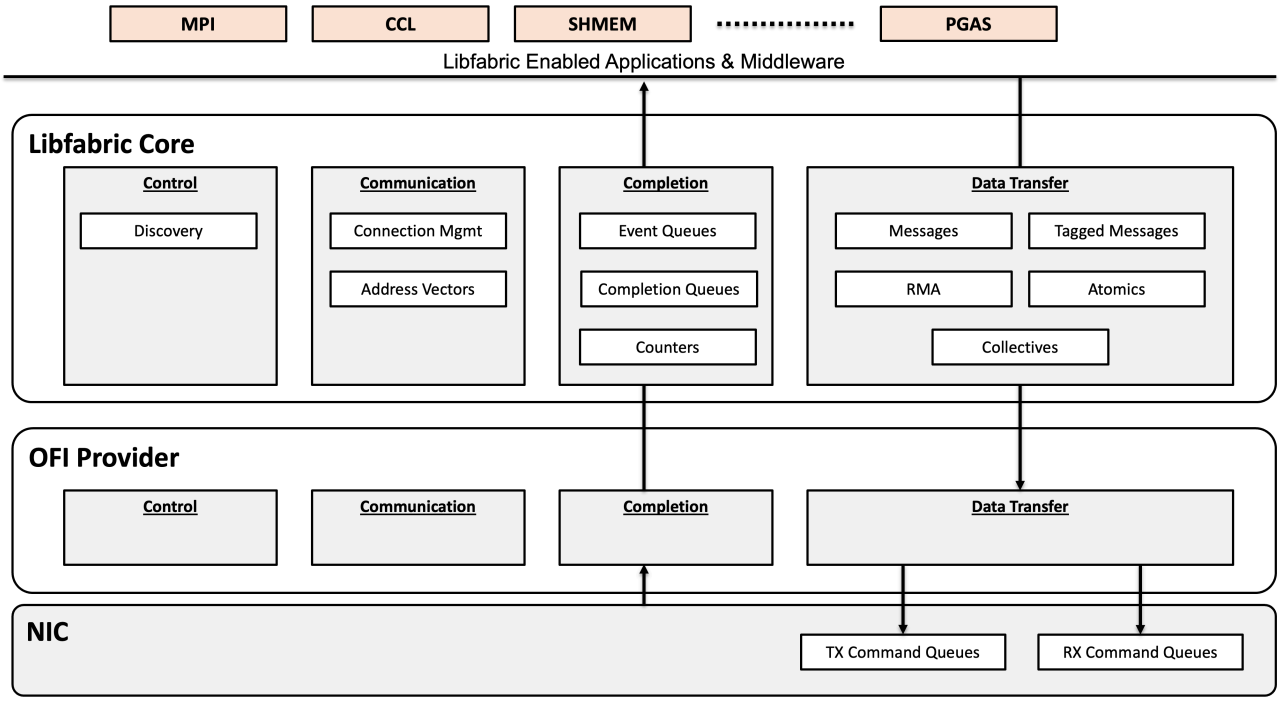


Figure 1. Libfabric architecture

As shown in Figure: Libfabric architecture , Libfabric provides an API for applications and middleware that require high-performance communication. Applications that require collective communication, such as those used in AI and HPC, can directly use libfabric's Collectives API, but since there are efficient high-level libraries (middleware) such as MPI, it is recommended that normal applications use them.

The architecture is designed to bypass the OS kernel to enable high-speed communication including low latency, but it is not required and you can choose whether to bypass it. For example, if you select tcp or udp for the OFI Provider, you can use the kernel's network stack.

Libfabric consists of Libfabric Core and OFI Provider. The role of Core is to mediate between applications or middleware and specific providers. Specifically, it provides APIs to applications, provides information on available providers, and passes processing to OFI Providers according to the called API.

To support a new transport, such as UET, implement code that processes the APIs defined in Core and add it as an OFI Provider.

Figure: In the Libfabric architecture , the functions provided by Libfabric are grouped by service. Also, implementations corresponding to the services and functions in Libfabric exist on the OFI Provider side.

Next, we will explain the functions provided by each service and the other components that make up Libfabric.

Control Service

The Control service provides a Discovery function that identifies the hardware available in the system, the features supported by the platform and operating system, and associated communication and computation libraries, and shares this information with applications.

Communication Services

The Communication service manages connections for communication between processes and provides the Connection Mgmt and Address Vectors functions.

In Libfabric, inter-process communication is performed between objects (structures) called endpoints, and supports both connection-oriented and connectionless communication. There are three communication models supported by Libfabric. (Reliability is a function of the endpoint's data transfer protocol.)

FI_EP_MSG - Reliable Connected

Reliable message communication (connection-oriented)

FI_EP_DGRAM - Unreliable Datagram

Unreliable data communication (connectionless)

FI_EP_RDM - Reliable Unconnected (Datagram)

Reliable data communication (connectionless)

The Connection Mgmt function manages connections in connection-oriented communication and endpoints in connectionless communication.

Address Vectors are used to improve scalability of connectionless communication and are explained in the Libfabric Object Model .

Completion service

The Completion service is used to notify the result of a submitted operation.

There are two notification methods: Queues and Counters, with the following differences:

Queues

Notify including details of completed operations.

Counters

It only reports the number of operations that have been completed.

There are two types of queues: Event Queues (EQ) and Completion Queues (CQ). EQ is used to notify control events such as connection establishment, and CQ is used to notify communication operation (data transfer) events. (In Libfabric, data transfer is performed asynchronously.)

For more information on Event Queues (EQs) / Completion Queues (CQs), see the Libfabric Object Model .

Data Transfer Service

Libfabric offers multiple data transfer semantics to accommodate different application requirements.

There are five sets of data transfer APIs: Messages, Tagged Messages, RMA (Remote Memory Access), Atomics, and Collectives.

Collectives are atomic operations between multiple peers, while others are one-to-one communications between an initiator and target pair.

Messages

- Data is sent and received while taking message boundaries into consideration.
- It behaves like a FIFO, and packets are placed (matched) in the receive buffer in the order they are received by the target.

Tagged Messages

- Like the Messages API, data is sent and received with message boundaries in mind.
- Unlike Messages, messages are placed into a buffer based on small tags (steering tags) that you assign to them.
- All message buffers, both sending and receiving, are associated with a tag.

RMA (Remote Memory Access)

- RMA allows an application to directly access the memory space of a target process (on another node).
- Specifically, it is possible to perform operations such as writing data to a specific memory area of the target process (write), reading a specific memory area of the target process (read) and saving it in a local buffer.
- RMA is sometimes referred to as RDMA, but RMA refers to remote memory access in general, whereas RDMA is a term tied to specific transport implementations such as InfiniBand, RoCE, and iWARP.
- For example, RMA can be used with TCP, which does not have hardware offload. Conversely, when using RDMA with a verbs provider, the transfer process, including the transport function, is offloaded to the NIC, and memory transfer is performed without the intervention of the processor (CPU).

Atomics

- Atomics adds arithmetic operations to RMA transfers, performing the operations simultaneously with direct access to the memory regions of the target process.
- Libfabric defines a variety of arithmetic operations that can be called as part of a data transfer operation.
- Examples include Minimum, Maximum, Sum, Product, etc. For a list, see "Atomic Operations" in man fi_atomic(3) [5].

Collectives

- Collectives, also known as collective operations, are atomic operations that occur between any number of peers.
- Examples include join, barrier, broadcast, allreduce, reduce_scatter, etc. See man fi_collective(3) [6] for a list.

OFI Provider

OFI Providers are implementations of various transports and hardware processing that correspond to the APIs provided by the Libfabric Core.

When an application calls a Libfabric API, the function call is routed directly to the provider for execution via a function pointer associated with the Libfabric object. This separation of the Libfabric Core and OFI Providers allows each provider to optimize the communication semantics defined by libfabric according to the network hardware, operating system, platform, and network protocol capabilities available to it.

OFI Provider consists of Core Providers that correspond to specific transports and network devices, and various providers that support them (Utility, Hooking, Offload, LINKx).

!! A reference implementation of UET will also be published as a new Core Provider. !!

The main Core Providers are listed below.

Verbs

- Provider of RDMA NICs, such as InfiniBand, RoCE, etc.
- Compatible with both Linux and Windows platforms.

SHM

- A provider of intra-node communication using shared memory.

EFA

- Provider of Amazon EC2 Elastic Fabric Adapter (EFA)[7].
- EFA is an OS-bypass hardware interface custom built by Amazon for EC2 instances that allows for improved performance of inter-instance communication.

CXI

- Provider of HPE (Cray) Slingshot[8].
- Slingshot makes some backwards-incompatible modifications to standard Ethernet to reduce latency, such as removing the preamble and the Interframe Gap.

UCX

- A provider of UCX libraries supported on NVIDIA's Infiniband fabric.
- UCX provides GPU-specific extensions (such as GPUDirect RDMA) to enable efficient data communication between GPU memories, and is also used as a backend for NCCL.

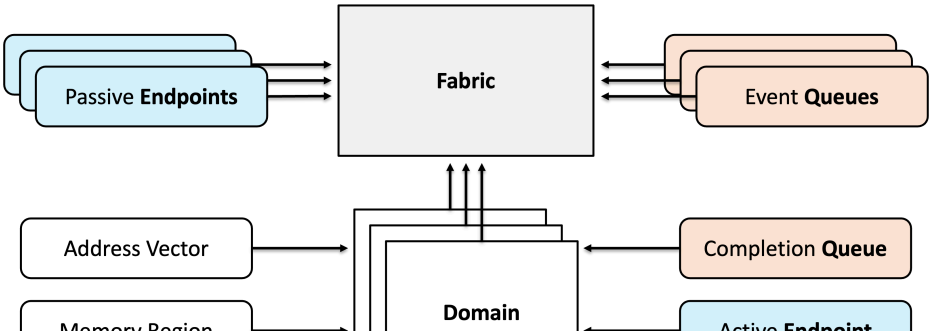
There are other providers such as LPP, OPX, PSM2, PSM3, TCP, and UDP. For details, see "Core Providers" in man fi_provider(7)[9].

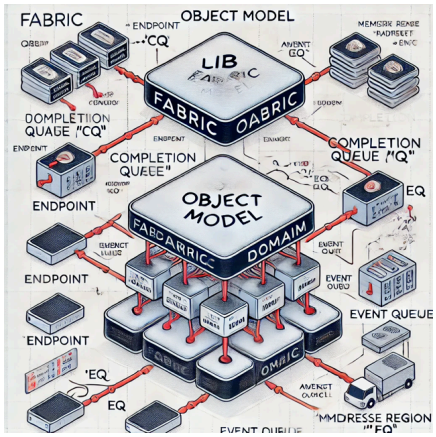
Libfabric Object Model

Libfabric is implemented in C, but it uses an object-oriented model and is written in a C++ style.

This section introduces the main object model that is necessary to understand the concepts of Libfabric.

(At first, I had ChatGPT 4o do the math, but it wasn't accurate, so I did it myself.)





Left: Libfabric Object Model: Created by the author with reference to man fi_arch(7)[10]
"Object Model" Right: Libfabric Object Model: Incorrect diagram created by ChatGPT 4o

From here on, we will explain each object with reference to man fi_arch(7)[11] "Object Model".

Fabric

- A Fabric is a collection of hardware and software resources accessible to a single physical or virtual network. It represents a single network (i.e. a subnet). All network ports that belong to the Fabric can communicate with each other through the Fabric.
- A Fabric is a top-level object to which other objects belong.

Domain

- A domain represents a logical entity connected to the fabric.
- In the simplest case this corresponds to one physical or virtual NIC, but it may involve multiple NICs (eg, multi-rail providers) or even no NICs, such as when using shared memory.
- A domain defines the boundaries within which resources are associated - for example, active endpoints and their corresponding completion queues must belong to the same domain.

Passive Endpoint / Active Endpoint

- An endpoint is conceptually similar to a TCP or UDP socket; processes use endpoints to transfer data.
- A passive endpoint is used by connection-oriented protocols to listen for connection requests.
  - They often map onto software constructs and may span multiple domains.
  - A Passive endpoint becomes an Active endpoint when it is connected to receive requests from an Active endpoint.
- An active endpoint implements a network protocol and initiates a connection.

Event Queues (EQs) / Completion Queues (CQs)

- EQ is used to notify control events such as connection establishment, and CQ is used to notify data transfer events (in Libfabric, data transfer is performed asynchronously).
- The reason for separating EQ and CQ, i.e. separating Control events from Data Transfer events, is performance. EQ is often implemented in software using OS constructs. Also, Control events usually occur during application initialization and are orders of magnitude less frequent than Data Transfer events during data transfer.

Table 1. Comparison of Event Queues (EQ) and Completion Queues (CQ)

	Event Queues (EQ)	Completion Queues (CQ)
the purpose	Control event notification	Notification of the completion status of communication operations
Notification Content	Control events such as connections, errors, and resource states	Completion events such as send, receive, and RDMA operations
Targeted operations	General control and system level notifications	Completion status of communication operations (data transfer)
Main functions	fi_eq_read, fi_eq_readerr	fi_cq_read, fi_cq_readerr
Usage scenarios	Monitor connections and capture errors	Confirmation of communication completion, data processing

Memory Region

- A Memory Region represents an application's local memory buffer.
- For a process to access a memory region on another node, the application must first construct the memory region and grant permission to the provider.
- When other processes access a Memory Region, they do so with the correct permissions (eg, access keys).
- Memory Regions are required for certain types of data transfer operations, such as RMA and Atomics operations.

Address Vectors

- Address vectors are used to improve scalability with connectionless endpoints.

- By mapping higher layer addresses, such as IP addresses and hostnames, to fabric-specific addresses, providers can reduce the memory required to maintain large address look-up tables and avoid costly address resolution and lookup operations during data transfer.

---

1. There has been some discussion about the need for other APIs, such as direct support for RDMA Verbs, but no specific direction has been announced at this time.
2. **https://ofiwg.github.io/libfabric/v2.0.0/man/fi_provider.7.html**
3. **https://ofiwg.github.io/libfabric/**
4. "Keynote: Networking for AI and HPC, and Ultra Ethernet", NANOG 92, Hugh Holbrook, Arisa Networks
5. **https://ofiwg.github.io/libfabric/main/man/fi_atomic.3.html**
6. **https://ofiwg.github.io/libfabric/main/man/fi_collective.3.html**
7. **https://aws.amazon.com/hpc/efa/**
8. **https://www.hpe.com/psnow/doc/a50002546enw**
9. **https://ofiwg.github.io/libfabric/main/man/fi_provider.7.html**
10. **https://ofiwg.github.io/libfabric/main/man/fi_arch.7.html**
11. **https://ofiwg.github.io/libfabric/main/man/fi_arch.7.html**

Last updated 2025-01-28 16:28:42 +0900