**An Introduction to Open Fabric Interfaces**

*Abstract: OFI are high-performance software interfaces that provide low-latency access to fabric hardware.*

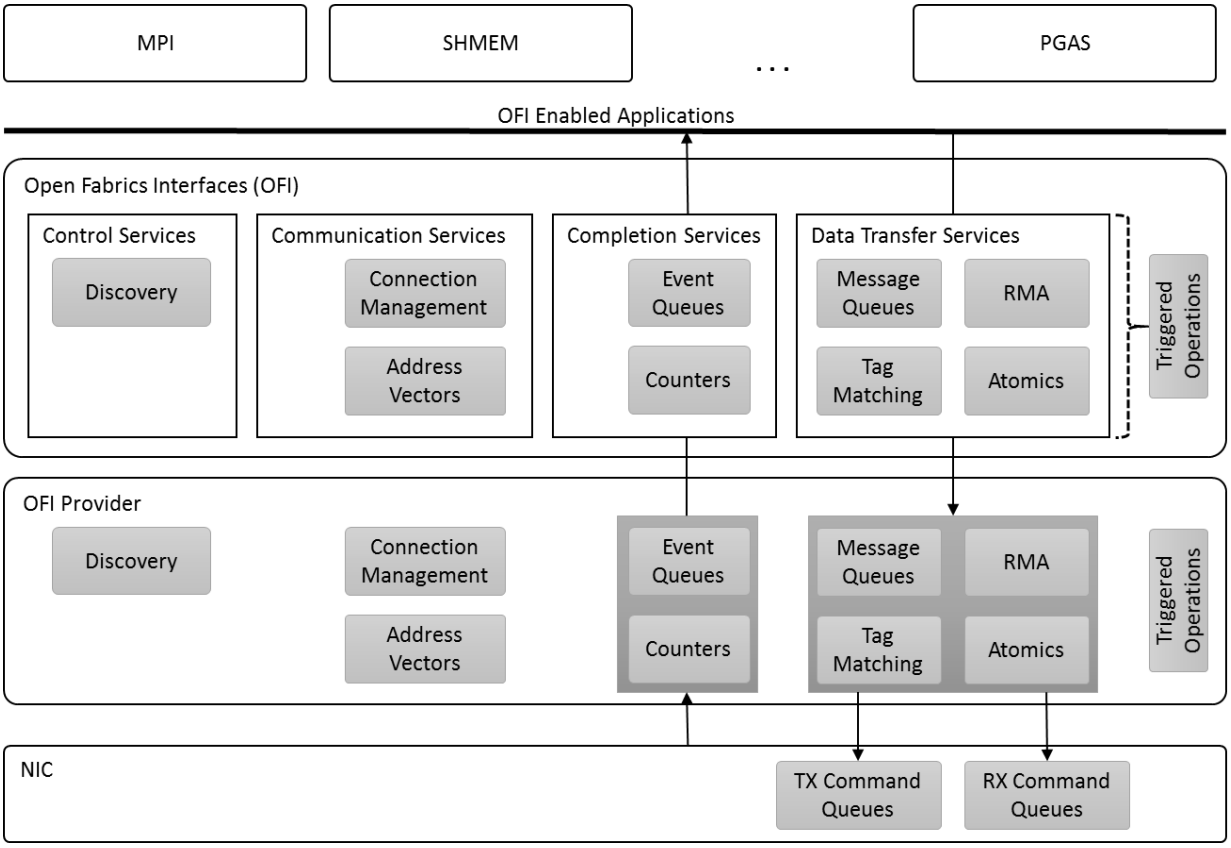
## Architectural Overview

Open Fabric Interfaces, or OFI, are a set of application programming interfaces (APIs) that export communication services to applications. OFI is specifically designed to meet the performance and scalability requirements of high-performance computing (HPC) applications, such as MPI, SHMEM, PGAS, DBMS, and enterprise applications, running in a tightly coupled network environment. OFI is agnostic to the underlying networking protocols, as well as the implementation of the networking devices.

OFI is instantiated in the libfabric library. Libfabric is supported on commonly available Linux based distributions.

OFI is architected to support process direct I/O. Process direct I/O, historically referred to as RDMA, allows an application to access network resources without operating system interventions. Data transfers can occur between networking hardware and application memory with minimal software overhead. Although OFI supports process direct I/O, it does not mandate any implementation or require operating system bypass.

The following diagram highlights the general architecture of the interfaces exposed by OFI. For reference, the diagram shows OFI in reference to a NIC. This is provided as an example only of how process direct I/O may be supported.




**OFI Architecture**

OFI is divided into two separate components. The main component is the OFI framework, which defines the interfaces that applications use. The OFI frameworks provides some generic services; however, the bulk of the OFI implementation resides in the providers. Providers plug into the framework and supply access to fabric hardware and services. Providers are often associated with a specific hardware device or NIC. Because of the structure of the OFI framework, applications access the provider implementation directly for most operations, in order to ensure the lowest possible software latencies.

As captured in the architecture diagram, OFI can be grouped into four main collections of interface sets.

Control Services: These are used by applications to discover information about the types of communication services are available in the system. For example, discovery will indicate what fabrics are reachable from the local node, and what sort of communication each provides.

Communication Services: These interfaces are used to setup communication between nodes. It includes calls to establish connections (connection management), as well as functionality used to address connectionless endpoints (address vectors).

Completion Services: OFI exports asynchronous interfaces. Completion services are used to report the results of submitted operations. Completions may be reported using event queues, which provide details about the operation that completed. Or, completions may be reported using lower-impact counters that simply return the number of operations that have completed.

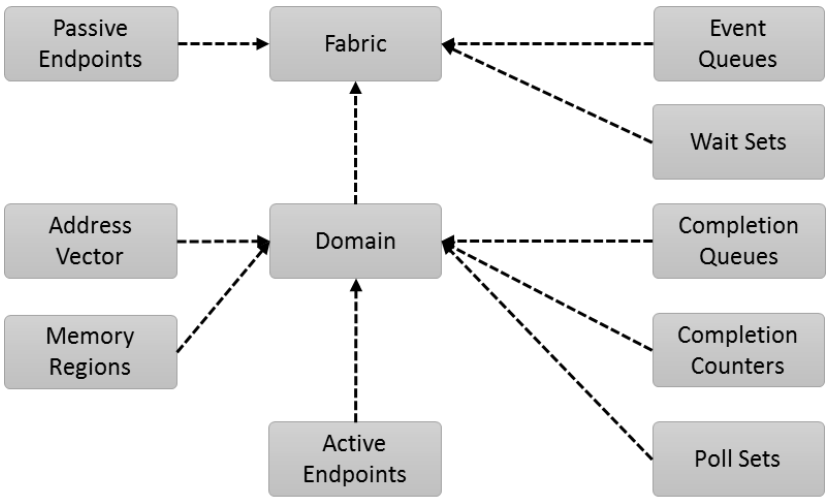
Data Transfer Services: These are sets of interfaces designed around different communication paradigms. Although shown outside the data transfer services, triggered operations are strongly related to the data transfer operations.

There are four basic data transfer interface sets. Message queues expose the ability to send and receive data with message boundaries being maintained. Message queues act as FIFOs, with sent messages matched with receive buffers in the order that messages are received. Tag matching is similar to message queues in that it maintains message boundaries. Tag matching differs from message queues in that received messages are directed into buffers based on small steering tags that are carried in the sent message.

RMA stands for remote memory access. RMA transfers allow an application to write data directly into a specific memory location in a target process, or to read memory from a specific address at the target process and return the data into a local buffer. Atomic operations are similar to RMA transfers, in that they allow direct access to the memory on the target process. Atomic operations allow for manipulation of the memory, such as incrementing the value found in memory. Because RMA and atomic operations provide direct access to a process's memory buffers, additional security synchronization is needed.

**Object Model**

Interfaces exposed by OFI are associated with different objects. The following diagram shows a high-level view of the parent-child relationships.




**OFI Parent-Child Object Relationships**

Fabric: A fabric represents a collection of hardware and software resources that access a single physical or virtual network. For example, a fabric may be a single network subnet. All network ports on a system that can communicate with each other through the fabric belong to the same fabric domain. A fabric shares network addresses and can span multiple providers.

Domain: A domain represents a logical connection into a fabric. For example, a domain may map to a physical or virtual NIC. A domain defines the boundary within which fabric resources may be associated. Each domain belongs to a single fabric.

Passive Endpoint: Passive endpoints are used by connection-oriented protocols to listen for incoming connection requests. Passive endpoints often map to software constructs and may span multiple domains.

Event Queues: EQs are used to collect and report the completion of asynchronous operations and events. Event queues handle events that are not directly associated with data transfer operations, referred to as control events. For example, connection requests and asynchronous errors that are not associated with a specific data transfer are reported using event queues.

Wait Sets: Although OFI allows the use of native operating system constructs for applications to use when waiting for events, it defines an optimized method for applications to use when waiting on events across multiple event queues, completion queues, and counters. Wait sets allow a single underlying wait object to be signaled whenever a specified condition occurs on an associated event queue, completion queue, or counter.

Active Endpoint: Active endpoints are data transfer communication portals. Active endpoints are used to perform data transfers, and are conceptually similar to a socket. Active endpoints are often associated with a single hardware NIC.

Completion Queue: Completion queues are high-performance queues used to report the completion of data transfer operations. Unlike fabric event queues, completion queues are often associated with a single hardware NIC.

Completion Counter: Completion counters are used to report the number of completed data transfer operations. Completion counters are considered lighter weight than completion queues, in that a completion simply increments a counter, rather than placing an entry into a queue.

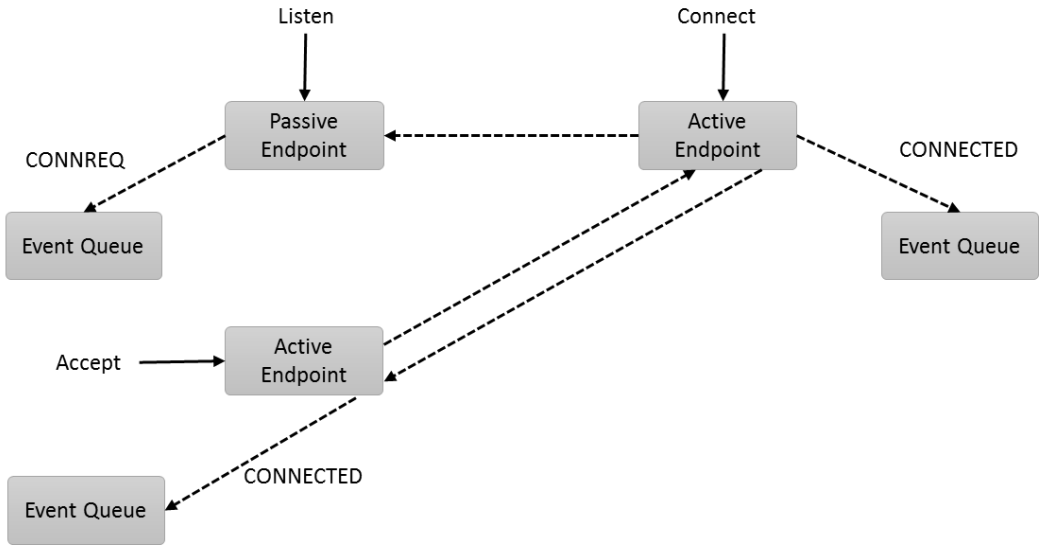
Poll Set: OFI allows providers to use an application's thread to process asynchronous requests. This can provide performance advantages for providers that use software to progress the state of a data transfer. Poll sets allow an application to group together multiple objects, such that progress can be driven across all associated data transfers. In general, poll sets are used to simplify applications where manual progress model is employed.

Memory Region: Memory regions describe application's local memory buffers. In order for fabric resources to access application memory, the application must first grant permission to the fabric provider by constructing a memory region. Memory regions are required for specific types of data transfer operations, such as RMA and atomic operations.

Address Vectors: Address vectors are used by connectionless endpoints. They map higher level addresses, such as IP addresses, which may be more natural for an application to use, into fabric specific addresses. The use of address vectors allows providers to reduce the amount of memory required to maintain large address look-up tables, and eliminate expensive address resolution and look-up methods during data transfer operations.

**Connection-Oriented Communication**

OFI supports both connected and connectionless communication. The following diagram highlights the general usage behind connection-oriented communication.




**Connected Endpoints**

Connections require the use of both passive and active endpoints. In order to establish a connection, an application must first create a passive endpoint and associate it with an event queue. The event queue will be used to report the connection management events. The application then calls listen on the passive endpoint. A single passive endpoint can be used to form multiple connections.

The connecting peer allocates an active endpoint, which is also associated with an event queue. Connect is called on the active endpoint, which results in sending a connection request (CONNREQ) message to the passive endpoint. The CONNREQ event is inserted into the passive endpoint's event queue, where the listening application can process it.

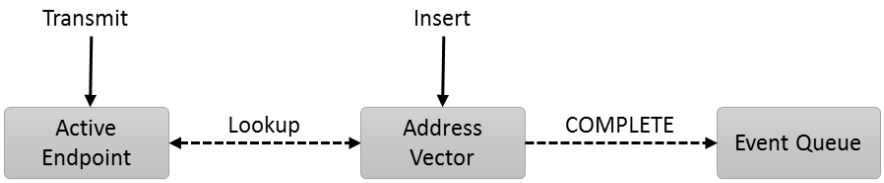
Upon processing the CONNREQ, the listening application will allocate an active endpoint to use with the connection. The active endpoint is bound with an event queue. Although the diagram shows the use of a separate event queue, the active endpoint may use the same event queue as used by the passive endpoint. Accept is called on the active endpoint to finish forming the connection. It should be noted that the OFI accept call is different than the accept call used by sockets. The differences result from OFI supporting process direct I/O.

OFI does not define the connection establishment protocol, but does support a traditional three-way handshake used by many technologies. After calling accept, a response is sent to the connecting active endpoint. That response generates a CONNECTED event on the remote event queue. If a three-way handshake is used, the remote endpoint will generate an acknowledgement message that will generate a CONNECTED event for the accepting endpoint. Regardless of the connection protocol, both the active

and passive sides of the connection will receive a CONNECTED event that signals that the connection has been established.

**Connectionless Communications**

Connectionless communication allows data transfers between active endpoints without going through a connection setup process. The diagram below shows the basic components needed to setup connectionless communication.



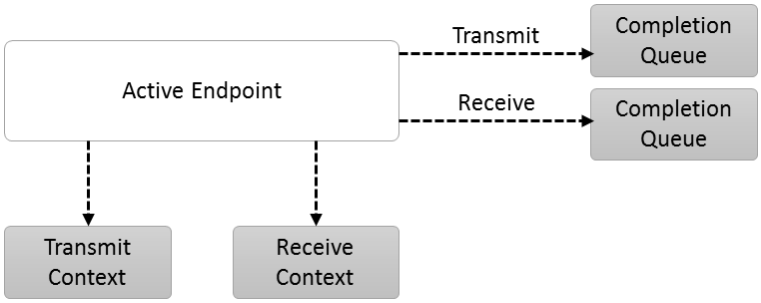

**Connectionless Endpoints**

OFI requires the addresses of peer endpoints be inserted into a local addressing table, or address vector, before data transfers can be initiated against the remote endpoint. Address vectors abstract fabric specific addressing requirements and avoid long queuing delays on data transfers when address resolution is needed. For example, IP addresses may need to be resolved into Ethernet MAC addresses. Address vectors allow this resolution to occur during application initialization time. OFI does not define how an address vector be implemented, only its conceptual model.

Address vectors may be associated with an event queue. After an address is inserted into an address vector and the fabric specific details have been resolved, a completion event is generated on the event queue. Data transfer operations against that address are then permissible on active endpoints that are associated with the address vector. Connectionless endpoints must be associated with an address vector.

**Endpoints**

Endpoints represent communication portals, and all data transfer operations are initiated on endpoints. OFI defines the conceptual model for how endpoints are exposed to applications, as demonstrated in the diagrams below.

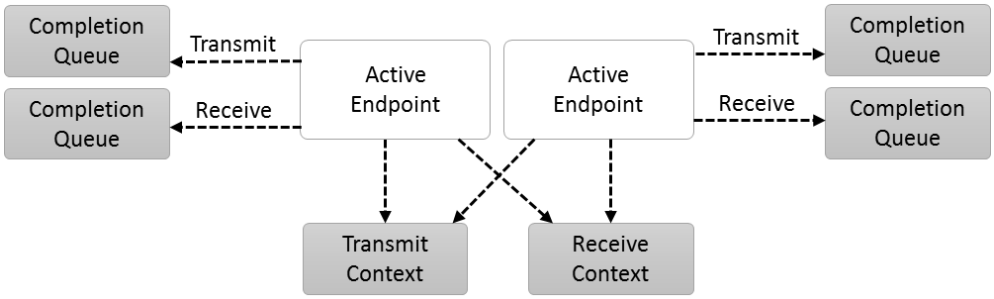



**Basic Endpoint Architecture**

Endpoints are usually associated with a transmit context and a receive context. Transmit and receive contexts are often implemented using hardware queues that are mapped directly into the process's address space, though OFI does not require this implementation. Although not shown, an endpoint may be configured only to transmit or receive data. Data transfer requests are converted by the underlying provider into commands that are inserted into transmit and/or receive contexts.

Endpoints are also associated with completion queues. Completion queues are used to report the completion of asynchronous data transfer operations. An endpoint may direct completed transmit and receive operations to separate completion queues, or the same queue (not shown).

A more advanced usage model of endpoints that allows for resource sharing is shown below.

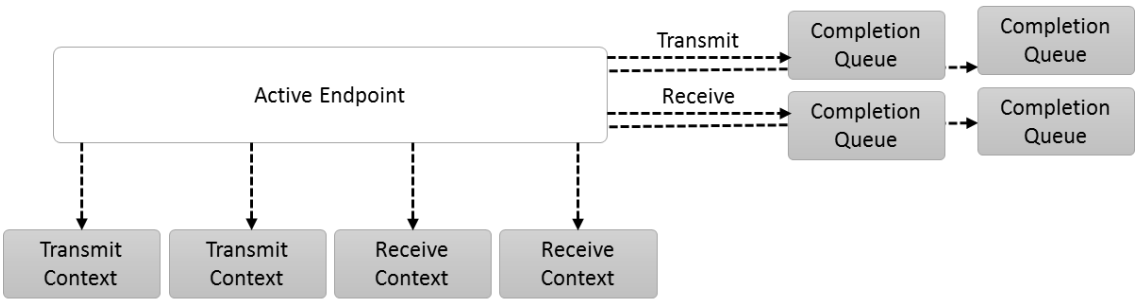



**Shared Transmit and Receive Contexts**

Because transmit and receive contexts may be associated with limited hardware resources, OFI defines mechanisms for sharing contexts among multiple endpoints. The diagram above shows two endpoints each sharing transmit and receive contexts. However, endpoints may share only the transmit context or only the receive context or neither. Shared contexts allow an application or resource manager to prioritize where resources are allocated and how shared hardware resources should be used.

Completions are still associated with the endpoints, with each endpoint being associated with their own completion queue(s).

The final endpoint model is known as a scalable endpoint. Scalable endpoints allow a single endpoint to take advantage of multiple underlying hardware resources.




**Scalable Endpoints**

Scalable endpoints have multiple transmit and/or receive contexts. Applications can direct data transfers to use a specific context, or the provider can select which context to use. Each context may be associated with its own completion queue. Scalable contexts allow applications to separate resources to avoid thread synchronization or data ordering restrictions.