Libfabric (OFI)

Table of Contents

UET　のリファレンス実装は、Libfabric　のプロバイダとして提供される予定です。[1] そのため、UETの技術スタックについて理解を深めるためには Libfabric や OFI の理解が必要となります。

なお、UECの理解のために Libfabric の全てを理解する必要はありません。 UEC技術の全体像や固有の技術に興味がある場合、Libfabricのアーキテクチャをざっと理解した上で、後で必要に応じて詳細を読んでいただいても大丈夫です。

参考資料

LibfabricやOFIはHPCの分野では広く利用されており、公式ドキュメントやカンファレンスの講演資料なども公開されています。 そのため、ChatGPTなどのAIに質問すると多くの参考資料を紹介してくれると思いますが、参考までに執筆にあたり参照した資料を例として列挙します。

- "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)` … Libfabric が開発された背景を詳細に解説
  - Architecture `fi_arch(7)` … Libfabric のアーキテクチャ解説（各コンポーネントの役割を含む）
- 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"

Libfabric と OFI, OFA の関係

一言で説明すると、OFIはAPI仕様、LibfabricはAPIの実装、OFAはそれらを策定し啓蒙している団体の名前です。（OFI と Libfabric API は同じ意味で使われる場合もあります）

OFA (OpenFabrics Alliance)

- 2004年に "OpenIB Alliance" という名前で設立され、Linuxで動作するベンダ中立なInfiniBandソフトウェアスタックの開発を目的としていました。 その後、Windowsのサポートや、プロトコルも2006年にiWarp、2010年にRoCEへの対応を行うなど拡張を続けました。 2014年にはOpenFabrics Interfaces working group を設立し、他の高性能ネットワーク技術への対応を進めています。
- 現在のビジョンはRDMAとカーネルバイパスのための、クロスプラットフォームでトランスポートに依存しないソフトウェアスタックを提供することです。 "トランスポートに依存しない" とは、アプリケーションが InfiniBand, iWARP, RoCE や他のファブリックプロトコルの上でも、それらを意識することなく動作するAPIを提供することです。

OFI (Open Fabric Interface)

- OFI は、高性能なネットワーク通信を効率的に行うために設計されたAPI仕様であり、アプリケーションがネットワークの詳細（ハードウェアやプロトコルの違い）に依存せずに通信を行える抽象化層を提供します。
- 具体的には、タグ付きメッセージの送受信（Tagged Messages）、リモートメモリアクセス（RMA）、アトミック操作（Atomics）、集団通信（Collectives）などのAPIを提供します。
- OFI のバージョンは 1.x と 2.0 があり、UET は OFI 2.0 をベースに設計されています。

Libfabric

- Libfabric は、OFIの実装を提供するライブラリであり、OFIのAPI仕様に基づいて開発されています。
- Libfabric はトランスポート技術に共通のAPIを提供し、"libfabric provider" と呼ばれるプラグインを追加することで多様なトランスポート技術をサポート可能なアーキテクチャとなっています。
- トランスポート技術の例としては、TCP, UDP, RDMA Verbs, EFA, SHM, そして UET などが挙げられます。

Libfabric のアーキテクチャ

OFIの実装である Libfabric のアーキテクチャ、アプリケーションに提供される各種サービス、その他関連するコンポーネントについて解説します。

なお、図：Libfabricのアーキテクチャ は、Libfabric Programmer’s Manual [3] や NANOG 92 Keynote 資料 [4] を参考に、著者が作成しました。

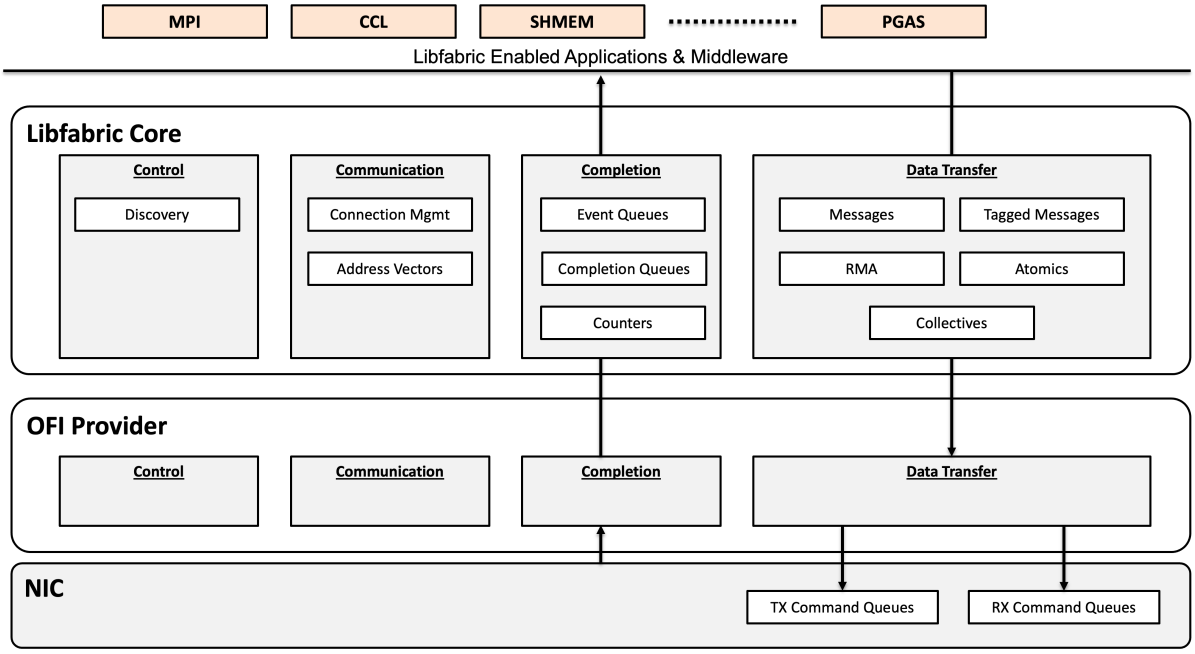


Figure 1. Libfabricのアーキテクチャ

図：Libfabricのアーキテクチャ のように、Libfabric は高性能通信が必要なアプリケーションやミドルウェアに対して API を提供します。 AIやHPCで利用される集団通信が必要なアプリケーションは、libfabric の Collectives API を直接利用することも可能ですが、MPIなどの効率の良い高レベルライブラリ（ミドルウェア）が存在しているため、通常のアプリケーションはそちらを利用することが推奨されます。

低遅延を含む高速な通信が可能なように、OSカーネルをバイパス可能なアーキテクチャになっていますが、必須ではなくバイパスするかは選択可能になっています。 例えば OFI Provider に tcp や udp を選択すると、カーネルのネットワークスタックを利用可能です。

Libfabric は Libfabric Core と OFI Provider から構成されます。 Core の役割は、アプリケーションやミドルウェアと、特定のプロバイダの間を仲介することです。 具体的には、アプリケーションに対しAPIを提供し、利用可能なプロバイダの情報を提供する、呼ばれたAPIに応じて OFI Provider に処理を渡す、などを行います。

UETなど、新しいトランスポートに対応するには、Core で定義されたAPIの処理を実行するコードを実装し、OFI Provider として追加します。

図：Libfabricのアーキテクチャ では、Libfabricで提供される機能をサービス毎にまとめてあります。 また、Libfabricに存在するサービスや機能に対応した実装が OFI Provider 側に存在しています。

以降、各サービス毎に提供される機能や、Libfabricを構成するその他コンポーネントについて解説します。

Control サービス

Control サービスは Discovery 機能を提供しています。 Discovery 機能は、システムで利用可能なハードウェア、プラットフォームやオペレーティングシステムでサポートされる機能、関連する通信ライブラリ、計算ライブラリを特定し、アプリケーションに情報を共有します。

Communication サービス

Communication サービスはプロセス間で通信を行うためのコネクション管理を行い、Connection Mgmt（コネクション管理）機能、Adress Vectors 機能を提供します。

Libfabricでは、プロセス間通信はエンドポイント（endpoint）というオブジェクト（構造体）の間で行われ、コネクションオリエンテッド通信、コネクションレス通信、両方に対応しています。 Libfabric がサポートする Communication Model は以下の3つです。（Reliability はエンドポイントのデータ転送プロトコルの機能です）

FI_EP_MSG - Reliable Connected

信頼性のあるメッセージ通信（コネクションオリエンテッド）

FI_EP_DGRAM - Unreliable Datagram

信頼性のないデータ通信（コネクションレス）

FI_EP_RDM - Reliable Unconnected (Datagram)

信頼性のあるデータ通信（コネクションレス）

Connection Mgmt機能は、コネクションオリエンテッド通信におけるコネクション管理や、コネクションレス通信におけるエンドポイントの管理を行います。

Address Vector は、コネクションレスな通信のスケーラビリティ向上のために利用されます。 Address Vector については Libfabric Object Model で解説します。

Completion サービス

Completion サービスは、submit された操作の結果を通知するために使用されます。

通知方法は Queues（キュー） と Counters（カウンター）の2つの方法があり、以下の違いがあります。

Queues

完了した操作の内容を含めて通知する。

Counters

完了した操作の数だけを通知する。

Queues は Event Queues (EQ) と Completion Queues (CQ) の2種類存在し、EQはコネクションの確立など制御（control）イベントの通知に利用され、CQは通信操作（data transfer）イベントの通知に利用されます。（Libfabricにおいて、データ転送は非同期に行われます）

Event Queues (EQs) / Completion Queues (CQs) の詳細は、Libfabric Object Model を参照してください。

Data Transfer サービス

Libfabric は、様々なアプリケーション要件に対応するため、複数のデータ転送セマンティクスを提供しています。

データ転送APIは5セットあり、Messages（メッセージ）、Tagged Messages（タグ付きメッセージ）、RMA（リモートメモリアクセス）、Atomics（アトミック）、Collectives（コレクティブ）、です。

Collectivesは複数ピア間の atomic operation で、それ以外は一対の initiator と target 間の1対1の通信です。

Messages

- メッセージ境界を意識したデータの送受信を行います。
- FIFOのように振る舞い、受信側（target）で受信した順番に受信バッファに配置（matched）されます。

Tagged Messages

- Messages API 同様、メッセージ境界を意識したデータの送受信を行います。
- Messages と異なり、メッセージに付与された小さなタグ（steering tags）に基づいてバッファに配置されます。
- メッセージバッファは送受信共に、全てタグと紐付けられています。

RMA (Remote Memory Access)

- RMA は、アプリケーションが直接（他ノードの）ターゲットプロセスのメモリ領域にアクセスすることを可能とします。
- 具体的には、ターゲットプロセスの特定のメモリ領域へデータ書き込む（write）、ターゲットプロセスの特定のメモリ領域を読み込み（read）ローカルバッファに保存する、等の操作を実行可能です。
- RMA は RDMA とも呼ばれることもありますが、RMAはリモートメモリアクセス全般を示すのに対し、RDMAは InfiniBand, RoCE, iWARP など特定のトランスポート実装に紐づいた用語として利用されています。
- 例えば、RMA はハードウェアオフロードの無い TCP 等でも利用可能です。逆に、verbs プロバイダを用いてRDMAを利用すると、NICにトランスポート機能含め転送処理がオフロードされ、プロセッサ（CPU）の介在無しにメモリ転送を行います。

Atomics

- Atomics は、RMA転送に算術演算を追加したものであり、ターゲットプロセスのメモリ領域への直接アクセスと演算を同時に実行します。
- libfabric では、データ転送操作の一部として呼び出し可能な様々な算術演算が定義されています。
- 具体例としては Minimum, Maximum, Sum, Product などが挙げられますが、一覧は man fi_atomic(3) [5] の "Atomic Operations" を参照してください。

Collectives

- Collectives は集団操作とも呼ばれ、任意の数のピア間で行われるアトミック操作です。
- 具体例としては join, barrier, broadcast, allreduce, reduce_scatter などが挙げられますが、一覧は man fi_collective(3) [6] を参照してください。

OFI Provider

OFI Provider は、Libfabric Core が提供するAPIに応じた、各種トランスポートやハードウェア処理の実装です。

アプリケーションが Libfabric API を呼ぶと、その関数呼び出しは Libfabric オブジェクトに関連付けられた関数ポインタ経由でプロバイダに直接ルーティングされ、実行されます。 このように Libfabric Core と OFI Provider を分離することにより、各プロバイダが使用可能なネットワークハードウェア、オペレーティングシステム、プラットフォーム、およびネットワークプロトコルの機能に応じて、libfabric で定義された通信セマンティクスの最適化が可能となります。

OFI Provider は特定のトランスポート及びネットワークデバイスに対応した Core Providers と、それをサポートする各種プロバイダ（Utility, Hooking, Offload, LINKx）から構成されます。

!! UETのリファレンス実装も、新しい Core Providers として公開される予定です。 !!

主な Core Providers を以下に挙げました。

Verbs

- InfiniBand、RoCE、等、RDMA NIC のプロバイダ。
- Linux, Windows両方のプラットフォームに対応。

SHM

- 共有メモリ（SHared Memory）を利用した、ノード内通信（intra-node）のプロバイダ。

EFA

- Amazon EC2 Elastic Fabric Adapter (EFA) [7] のプロバイダ。
- EFA は、EC2インスタンス用に Amazon によりカスタムビルドされた、インスタンス間通信（inter-instance）のパフォーマンス向上が可能なOSバイパスハードウェアインターフェイス。

CXI

- HPE(Cray) Slingshot [8] のプロバイダ。
- Slingshot は、プリアンブルや Interframe Gap を削除するなど、レイテンシ削減のために標準イーサネットへ後方互換性の無い改良を加えている。

UCX

- NVIDIA の Infiniband ファブリックでサポートされる、UCXライブラリのプロバイダ。
- UCX は、GPU に特化した拡張機能（GPUDirect RDMA など）を提供し、GPU メモリ間の効率的なデータ通信を可能にし、NCCLのバックエンドとしても利用されている。

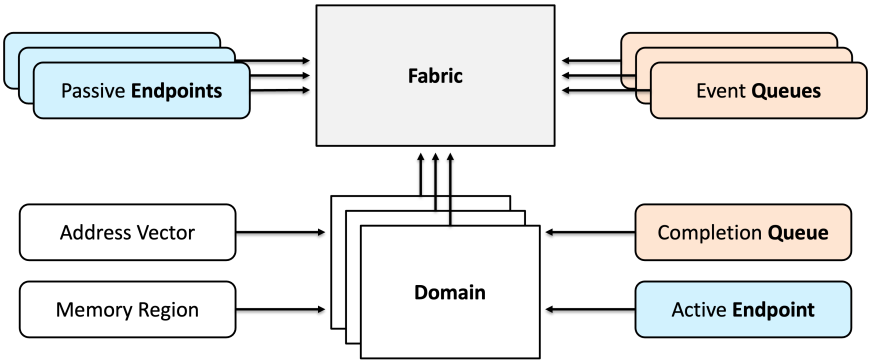
他にも LPP, OPX, PSM2, PSM3, TCP, UDP 等がありますが、詳細は man fi_provider(7) [9] の "Core Providers" を参照してください。

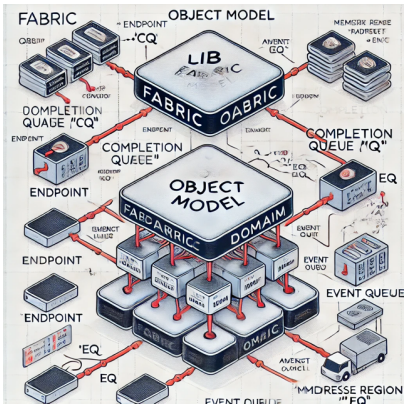
Libfabric Object Model

Libfabric はC言語で実装されていますが、オブジェクト指向モデルを採用し、C++的なプログラミングがされています。

Libfabric の概念を理解するにあたり必要となる主なオブジェクトモデルを紹介します。

（最初は ChatGPT 4o に錬成してもらいましたが、正しくないので著者が自分で作図しました）





左：Libfabric Object Model: man fi_arch(7) [10] "Object Model" を参考に著者が作成
右：Libfabric Object Model: ChatGPT 4o が錬成した誤った図

以降、man fi_arch(7) [11] "Object Model" を参考に各オブジェクトについて解説します。

Fabric（ファブリック）

- Fabric は、ひとつの物理または仮想ネットワークにアクセス可能なハードウェアやソフトウェア資源（resources）の集合体で、ひとつのネットワーク（例：サブネット）を表し、そのファブリックに所属するネットワークポートはファブリックを通じて相互に通信可能です。
- ファブリックは他のオブジェクトが属する、トップレベルのオブジェクトです。

Domain（ドメイン）

- ドメインはファブリックに接続された論理的な繋がりを表します。
- シンプルなケースでは1つの物理または仮装NICに対応しますが、複数のNICを含む場合もあります（e.g., multi-rail provider）が、共有メモリを使用する場合など、NICを含まない場合もあります。
- ドメインはリソース同士が関連付けられる境界を定義します。例えば、アクティブなエンドポイント（active endpoints）とそれに対応した completion queue は同じドメインに所属している必要があります。

Passive Endpoint / Active Endpoint（パッシブエンドポイント / アクティブエンドポイント）

- Endpoint は、概念的にはTCPやUDPソケットに似ており、プロセスがデータ転送を行う際はエンドポイントを使用します。
- Passive endpoint は、コネクション指向のプロトコルが接続要求を listen するために使用します。
  - 多くの場合、ソフトウェアの構造体にマップされ、複数のドメインにまたがることがあります。
  - Passive endpoint は Active endpoint からのリクエストを受信に接続すると、Active endpoint となります。
- Active endpoint はネットワークプロトコルを実装し、コネクションを開始します。

Event Queues (EQs) / Completion Queues (CQs)

- EQはコネクションの確立など制御（control）イベントの通知に利用され、CQは通信操作（data transfer）イベントの通知に利用されます。（Libfabricにおいて、データ転送は非同期に行われます）
- EQとCQを分離している、即ち Control events を Data Transfer events から分離しているのは、

性能が理由です。EQは多くの場合、OSの構造体を利用しソフトウェアで実装されています。また、Control events は通常アプリケーション初期化時に発生し、データ転送時は Data Transfer events と比較し数桁すくなく、ごく稀にしか発生しません。

Table 1. Event Queues (EQ) と Completion Queues (CQ) の比較

	Event Queues (EQ)	**Completion Queues (CQ)**
目的	制御イベントの通知	通信操作の完了状態の通知
通知内容	接続、エラー、リソース状態などの制御イベント	送信、受信、RDMA 操作などの完了イベント
対象となる操作	制御やシステムレベルの通知全般	通信操作（データ転送）の完了状況
主な関数	fi_eq_read, fi_eq_readerr	fi_cq_read, fi_cq_readerr
使用場面	接続の監視、エラーの取得	通信の完了確認、データ処理

Memory Region（メモリ領域）

- Memory Region はアプリケーションのローカル・メモリ・バッファを表します。
- プロセスが他のノードのメモリ領域にアクセスするためには、まずアプリケーションがメモリ領域を構築し、プロバイダに許可を与える必要があります。
- 他のプロセスが Memory Region にアクセスする際には、正しいパーミッション（e.g., access key）を用いてアクセスします。
- Memory Region は、RMAやAtomics操作など、特定のタイプのデータ転送操作に必要です。

Address Vectors（アドレスベクター）

- Address vectors は、コネクションレスな endpoints でスケーラビリティ向上のために利用されます。
- IPアドレスやホスト名といった高レイヤーのアドレスをファブリック固有のアドレスにマップすることにより、プロバイダは大きな address look-up tables を維持するのに必要なメモリを減らし、アドレス解決や検索といったコスト高な処理をデータ転送中に避けることが可能になります。

[1] RDMA Verbs の直接サポートなど、他のAPI対応の必要性に関する議論もありますが、現時点で具体的な方針は発表されていません。

[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