2025 OFA Webinar Series

LIBFABRIC: GETTING STARTED WITH ULTRA ETHERNET

Jianxin Xiong

Intel Corporation

MEET THE PRESENTER: JIANXIN XIONG, INTEL

Jianxin Xiong, Intel

Jianxin Xiong is a Principal Engineer at Intel. He has rich experience in various communication software stacks for HPC and AI, from MPI/CCL down to kernel RDMA driver. He is currently the chair of Open Fabrics Interface Working Group and the maintainer of Libfabric (https://ofiwg.github.io/libfabric). He is an active participant of several UEC Work Groups, including the Software and Transport Work Groups.

THE NEWS

UEC Launches Spec 1.0 Transforming Ethernet for AI and HPC at Scale

MAEMALYNN MEANOR | 11 JUNE 2025

Ultra Ethernet Consortium publishes 1.0 specification, readies Ethernet for HPC, AI

Ultra Ethernet Consortium | Jun 11, 2025

Ultra Ethernet Consortium launches 1.0 specification

Delivers a scalable solution across all layers of the networking stack

EXECUTIVE SUMMARY

First look of
Ultra Ethernet

what's the connection ?

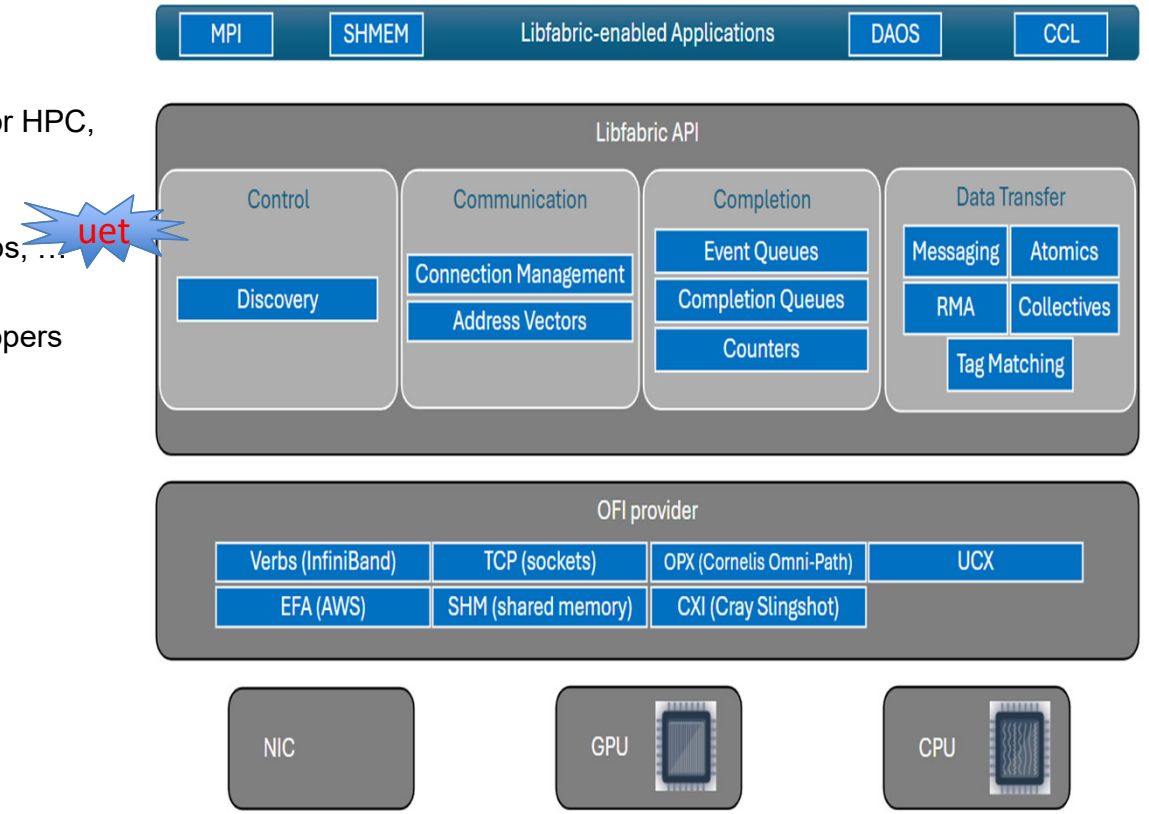
From the eyes of a libfabric developer

OUTLINE

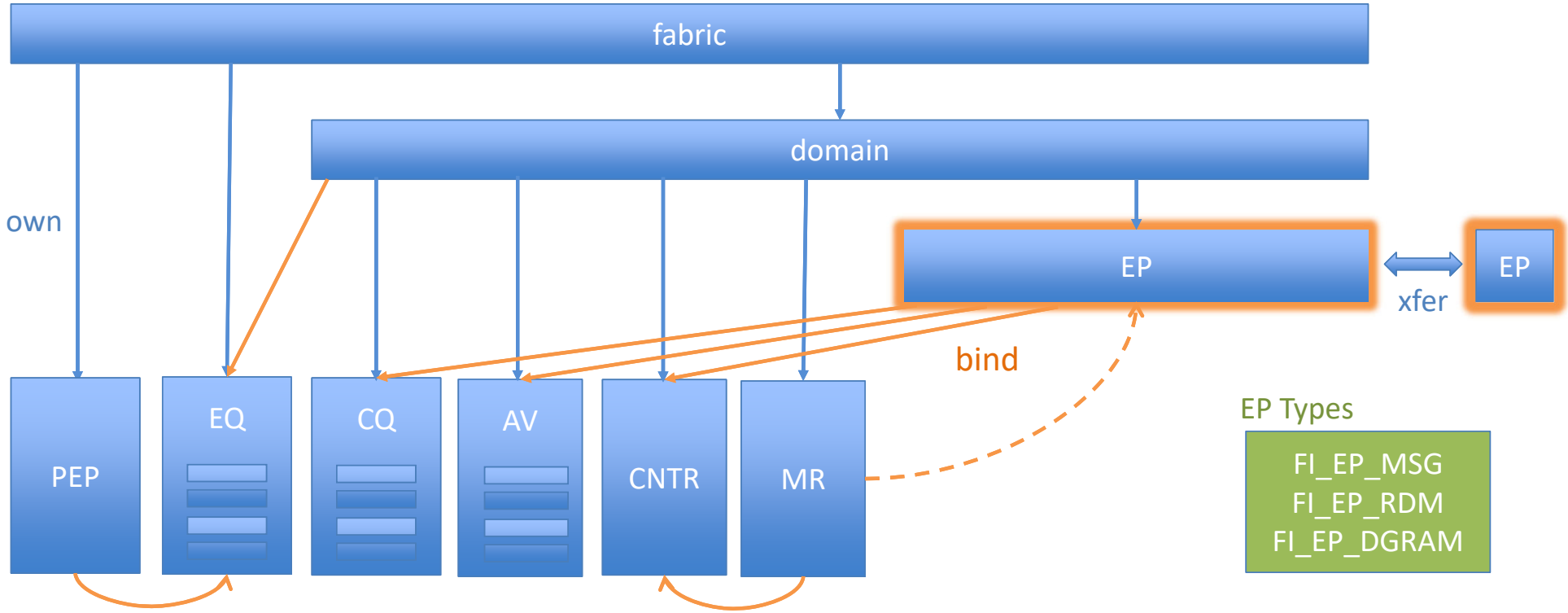
- Libfabric in A Nutshell
- Ultra Ethernet Overview
- Working with Libfabric over Ultra Ethernet

LIBFABRIC IN A NUTSHELL

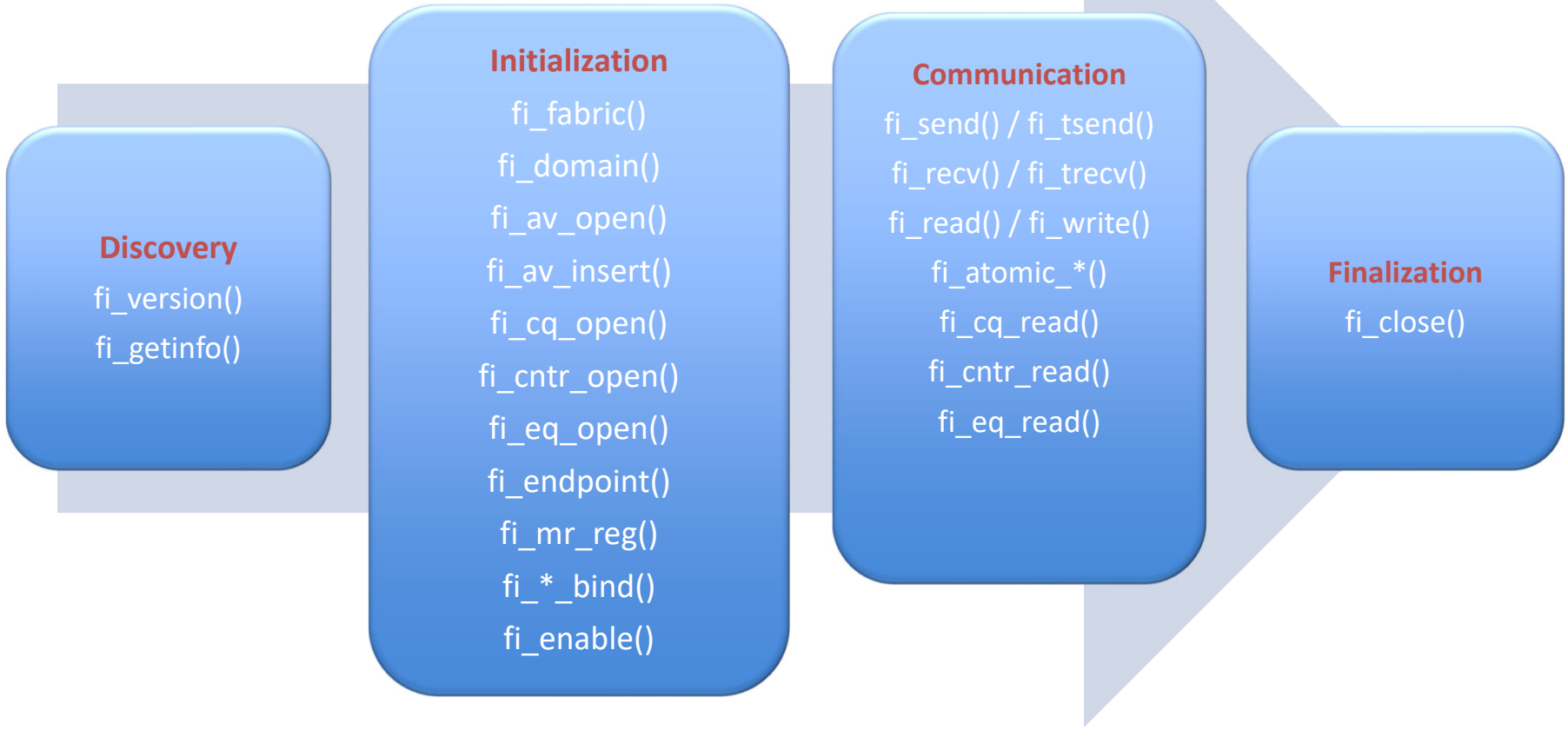
- **What is libfabric?**
  - A.K.A. OpenFabrics Interface (OFI), a project under OpenFabrics Alliance (OFA)
  - Community driven, low-level communication library for HPC, AI, and distributed storage
  - Abstract diverse networking technologies
    - Core providers: cxi, efa, opx, shm, tcp, ucx, verbs, …
    - Utility providers: hook, lnx, mrail, rxm, rxd, …
  - API co-designed with fabric providers and app developers
    - Minimize impedance mismatching
  - Queue based asynchronous RDMA operations
  - Support GPU/Accelerator memory (HMEM)
- **History of libfabric**
  - Project started: 2013
  - First stable release: v1.0.0, Apr 2016
  - Latest release: v2.1.0, March 2025
  - Upcoming release: v2.2.0, June 30, 2025
- **Links at the end**



LIBFABRIC OBJECTS



LIFE CYCLE OF A LIBFABRIC APPLICATION



OUTLINE

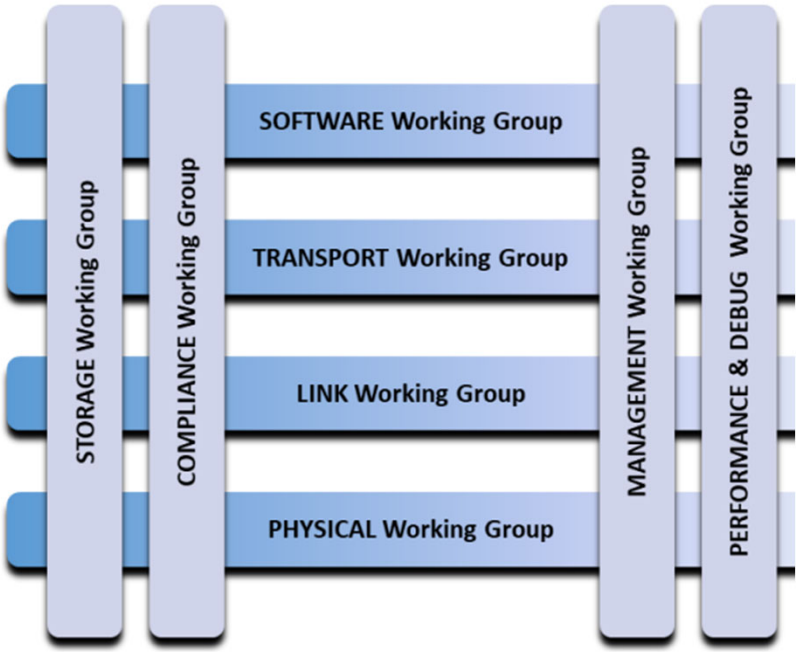
Libfabric in A Nutshell

Ultra Ethernet Overview

Working with Libfabric over Ultra Ethernet

WHAT IS ULTRA ETHERNET?

- **A high-performance Ethernet stack for HPC & AI**
  - Ethernet is ubiquitous
  - Evolve Ethernet to embrace new level of capability
  - Standard, not proprietary
  - Meet the demands from hyperscale computing and AI
    - Scale to 1M simultaneous endpoints
  - Maximizing performance, scalability, and efficiency
- **Ultra Ethernet Consortium (UEC)**
  - A consensus-based standards organization
  - Operating under the Linux Foundation
  - 90+ members as of today
  - 8 Working groups
  - https://ultraethernet.org
- **Ultra Ethernet Specification v1.0**
  - Released on 6/11/2025
  - https://ultraethernet.org/uec-1-0-spec

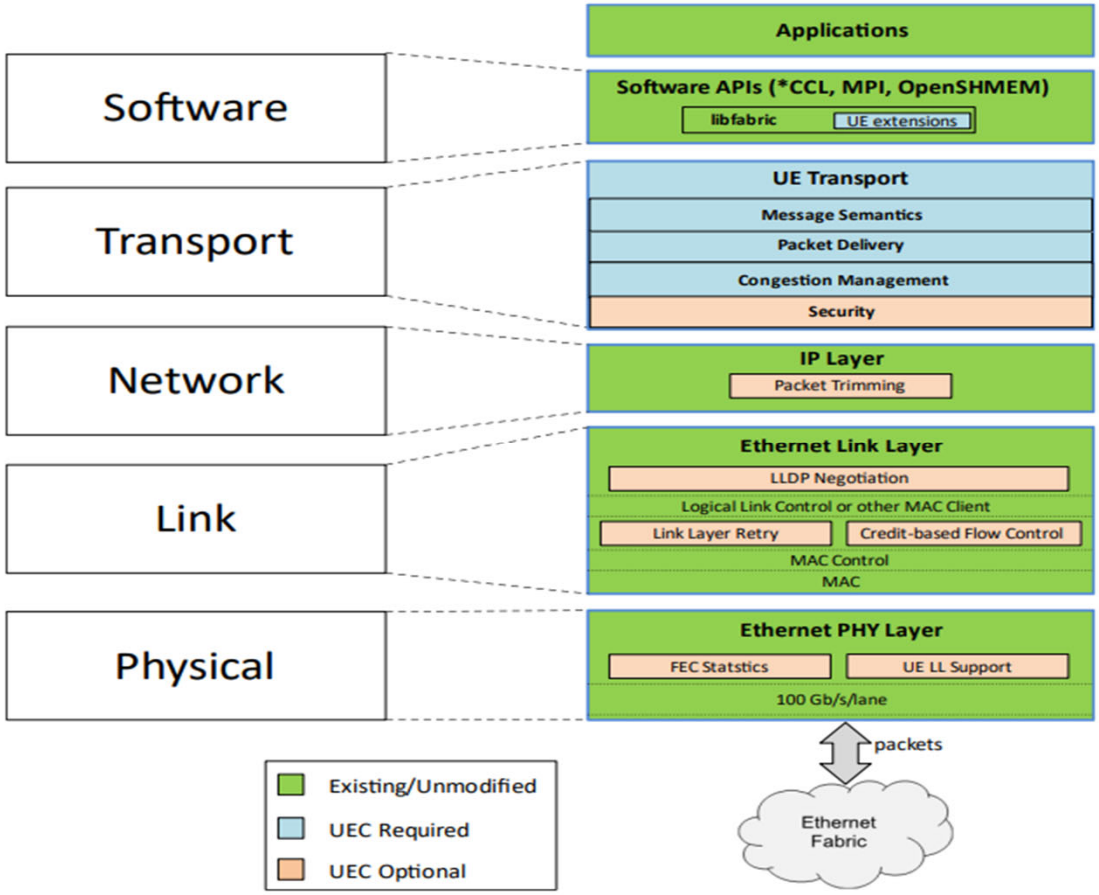


BENEFITS OF ULTRA ETHERNET NETWORKING

Traditional RDMA-Based Networking	Ultra Ethernet Consortium
Required In-Order Delivery, Go-Back-*N* recovery	Out-of-Order packet delivery with In-Order Message Completion
Security external to specification	Built-in high-scale, modern security
Flow-level multi-pathing	Packet Spraying (packet-level multipathing)
DC-QCN, Timely, DCTCP, Swift	Sender- and Receiver-based Congestion Control
Rigid networking architecture for network tuning	Semantic-level configuration of workload tuning
Scale to low tens of thousands of simultaneous endpoints	Targeting scale of 1M simultaneous endpoints

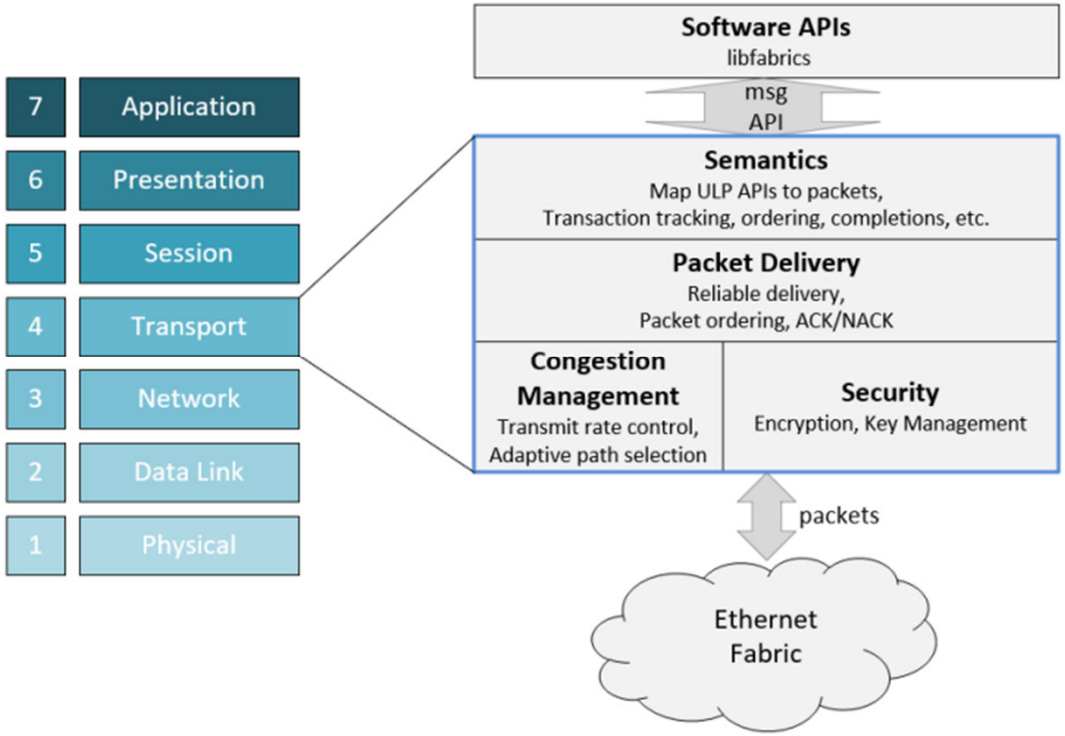
ULTRA ETHERNET SPECIFICATION LAYERS

- **Software layer**
  - UE libfabric mapping
- **Transport layer**
  - Semantics (SES)
  - Packet delivery (PDS)
  - Congestion management (CMS)
  - Security (TSS)
- **Network layer**
  - Packet trimming
- **Link layer**
  - Link layer retry (LLR)
  - Credit-based flow control (CBFC)
  - LLDP negotiation
- **Physical layer**
  - FEC statistics
  - UE LL support

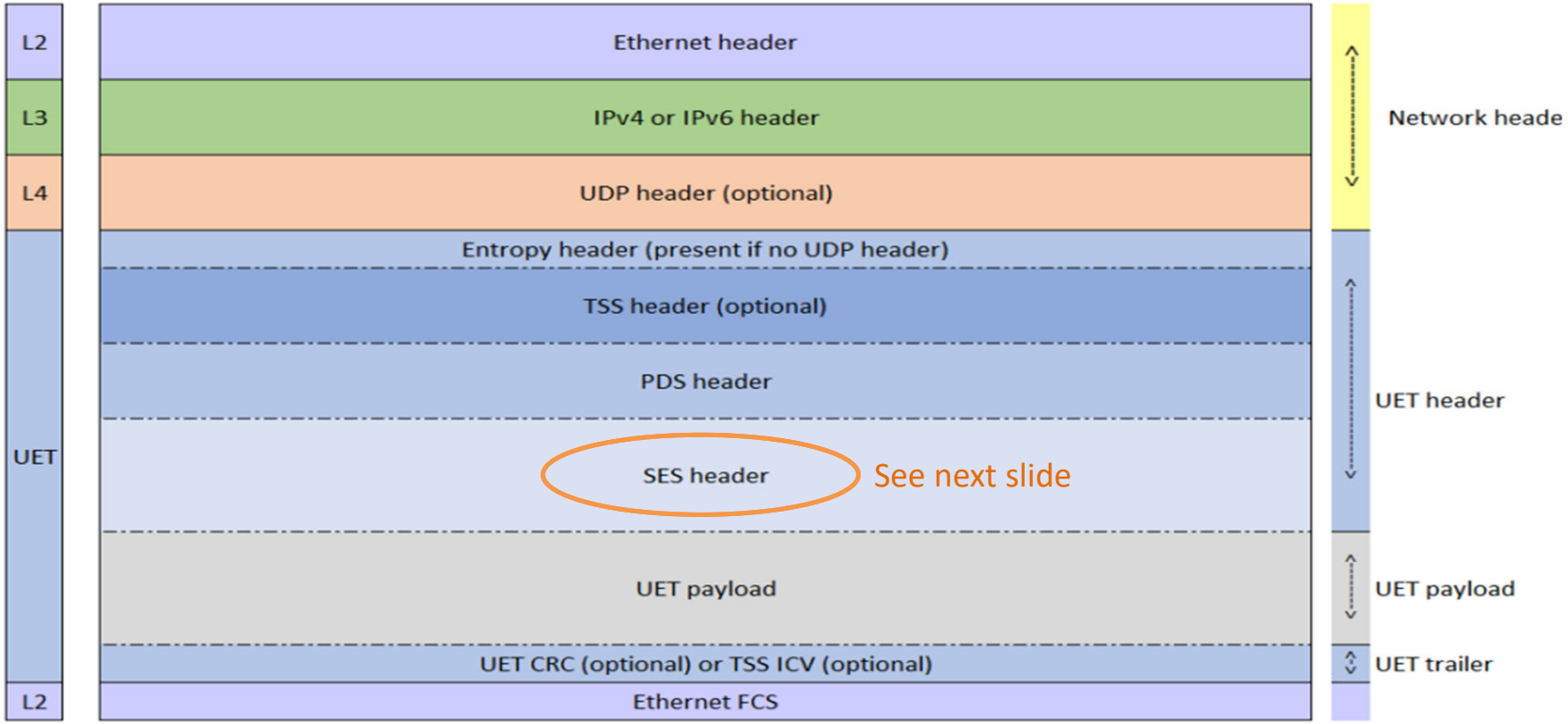


ULTRA ETHERNET TRANSPORT (UET)

- **The transport layer protocols are what make UE "UE"**
  - Designed to serve both HPC and AI workloads
    - Connectionless
    - Support RDMA
    - Provide a variety of communication primitives
      - sends (tagged / untagged)
      - deferrable sends (tagged / untagged)
      - rendez-vous sends (tagged / untagged)
      - writes / reads
      - atomics
    - JOB ID based authorization
    - Closely matches libfabric semantics
  - Multiple packet delivery ordering model
    - ROD, RUD, RUDI, UUD
  - Congestion control
    - NSCC & RCCC
  - Optional Security layer
- **Profiles to simplify implementation**
  - AI Base, AI Full, HPC

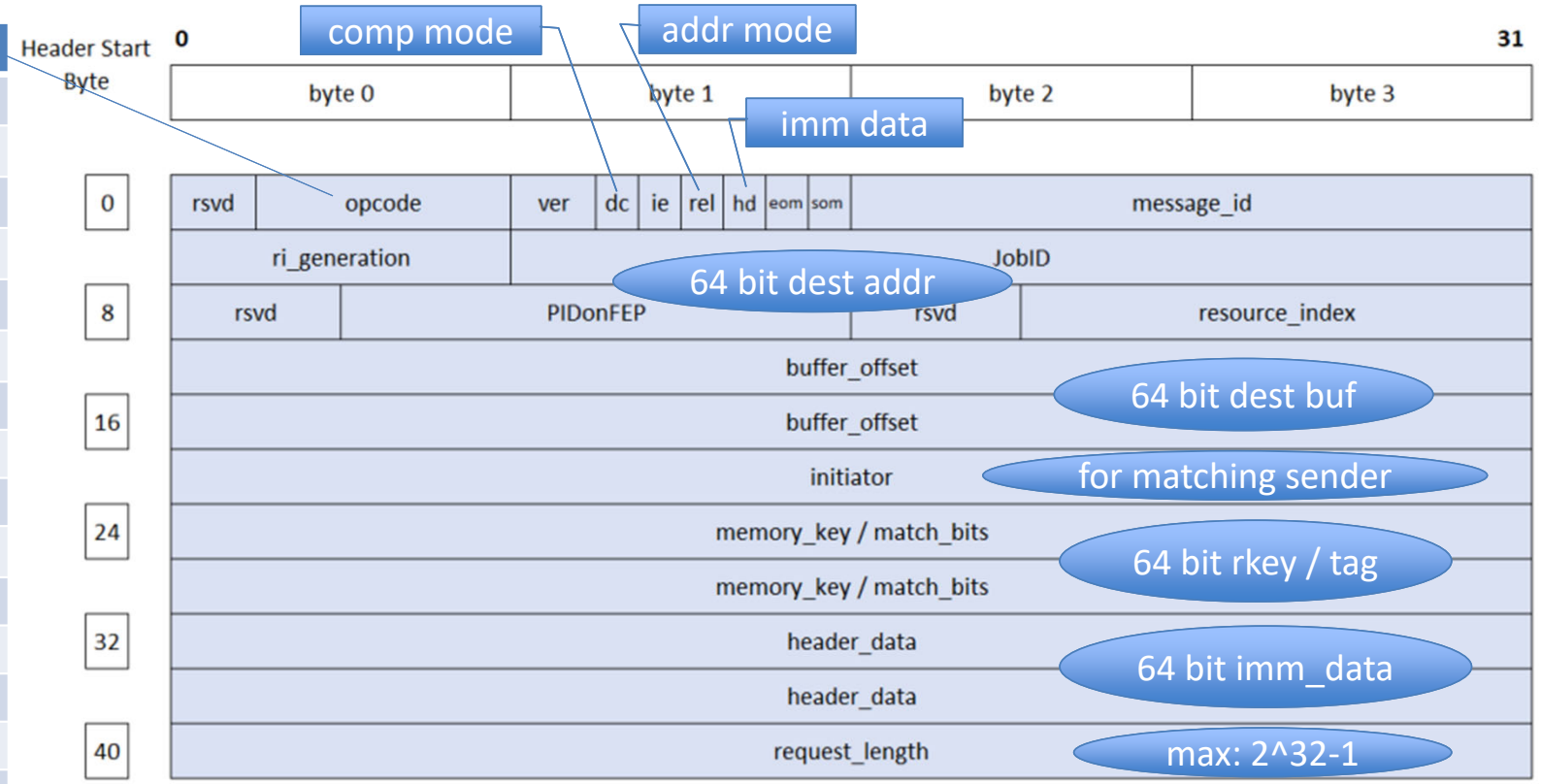


UET PACKET FORMAT



UET SES: STANDARD HEADER (44 BYTES)

Opcode (req)
UET_NO_OP
UET_WRITE
UET_READ
UET_ATOMIC
UET_FETCH_ATOMIC
UET_SEND
UET_RENDEZVOUS_SEND
UET_DATAGRAM_SEND
UET_DEFERRABLE_SEND
UET_TAGGED_SEND
UET_RENDZVOUS_TSEND
UET_DEFFERABLE_TSEND
UET_DEFFERABLE_RTR
UET_TSEND_ATOMIC
UET_TSEND_FETCH_ATOMIC
……



Other headers: **optimized** (non tag, no data), **rndv-ext**, **atomic-ext**, **response**, etc.
Determined by: **pds.next_hdr**, **ses.opcode**

UET DIFFERENT SEND OPS

- **Send (tagged / untagged)**
  - "Eager send":
    - Data is carried in the packet payload
    - Deliver to the receive buffer if matching buffer is found
    - Handled as "unexpected message" is no matching buffer is found

for small messages

- **Rendezvous Send (tagged / untagged)**
  - Request to send
    - Carry initiator buffer information (addr, key, etc)
    - Can optionally carry a portion of "eager data"
  - Target side issues Read when matching receive buffer is found
  - Target side send ACK when read is done

for large messages

- **Deferrable Send (tagged / untagged)**
  - "Flow controlled" by the target based on the readiness of receive buffer
  - Based on the ACKs from the target, send can proceed normally, or be deferred, or restart (from certain offset)
  - Less overhead than RNDV Send in "expected" case

for large messages

UET ADDRESSING

- **How to select a target?**
  - Fabric Address (FA): Network IP address of the Fabric Endpoint (FEP)
  - PIDonFEP: Logical index within the FEP, identifying a set of resources associated with a process
  - Job ID: Global unique identifier assigned to the job
  - Resource Index (RI): Logical index identifying a “service” (e.g. MPI, CCL) within the process
  - **Relative Addressing**:
    - Local PIDonFEP table per Job ID
    - Target determined by <FA, JobID, PIDonFEP, RI>
  - **Absolute Addressing**:
    - global PIDonFEP table
    - Target determined by <FA, PIDonFEP, RI>
    - JobID is used for authorization
- **How to identify a buffer at the selected target?**
  - Tag (ses.match_bits) for tagged recv or memory key (ses.memory_key) for RMA
  - Initiator ID (ses.initiator) if FI_DIRECTED_RECV is enabled
  - Must be authorized with Job ID

UET PROFILES

- **UET covers a wide range of semantics and capabilities**
- **Applications have different requirements, often only need a subset**
- **Profiles allow specialized implementation to simplify and optimize**
- **Three profiles**
  - **AI Base**: support *CCL and UD
  - **AI Full**: all AI training & AI inferencing applications
  - **HPC**: full-fledged HPC semantics, wide range of applications
- **Profiles are negotiated at initialization**
  - Part of libfabric endpoint address
  - AI Full is not a subset of HPC, but can be used as if it is with some restrictions (deferred treated as regular)

	AI BASE	AI FULL	HPC
NO_OP	MUST	MUST	MUST
SEND	1 MTU	4GB-1	4GB-1
DATAGRAM SEND	MUST	MUST	MUST
TSEND (exact match)	MUST	MUST	MUST
TSEND (wildcard)		MUST	MUST
WRITE / WRITE IMM	MUST	MUST	MUST
READ		MUST	MUST
Non-fetching Atomic	MUST	MUST	MUST
Fetching Atomic		MUST	MUST
Tagged Atomics			MUST
Deferrable SEND		MUST	
Deferrable TSEND		MUST	
RENDEZVOUS			MUST

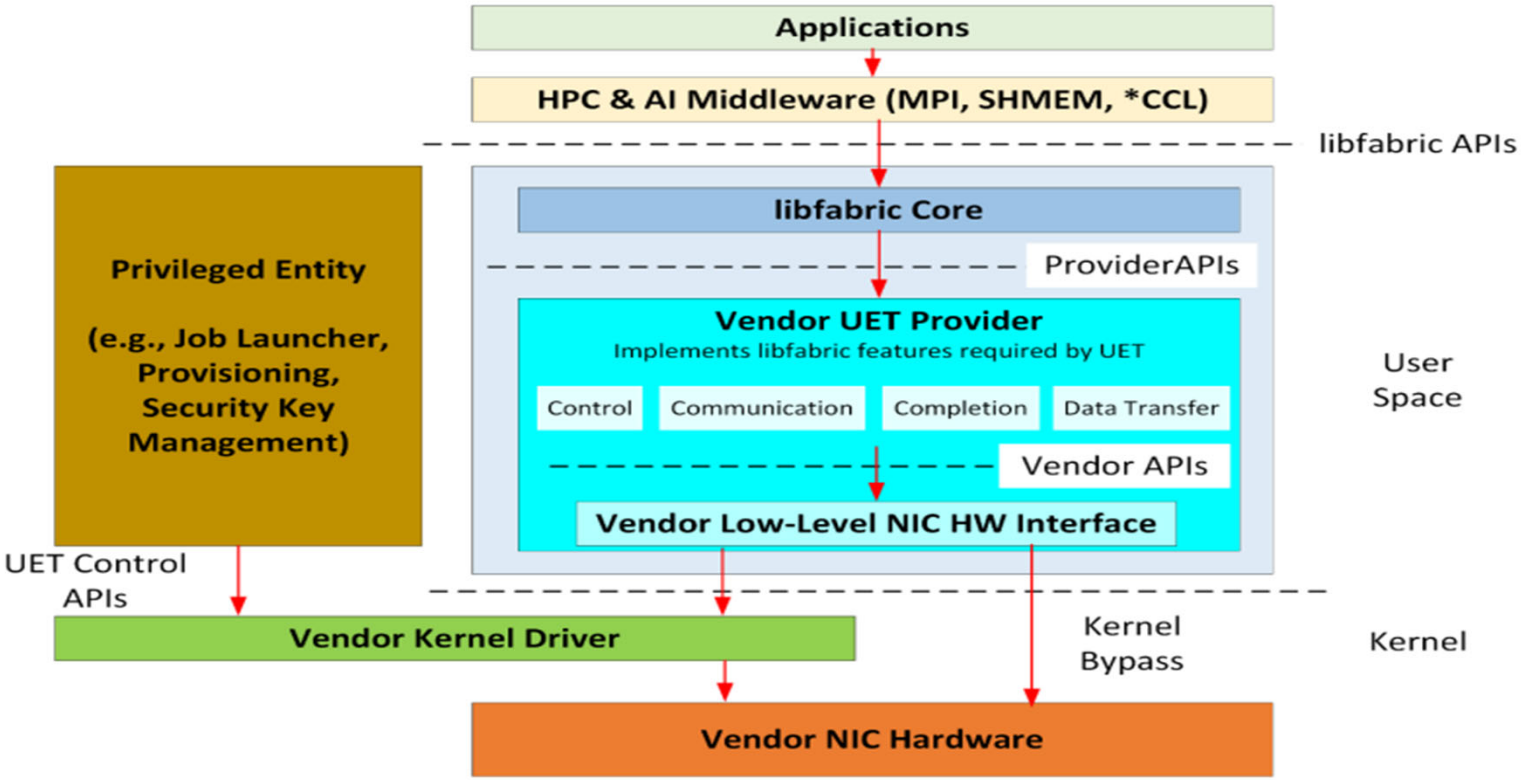
OUTLINE

Libfabric in A Nutshell

Ultra Ethernet Overview

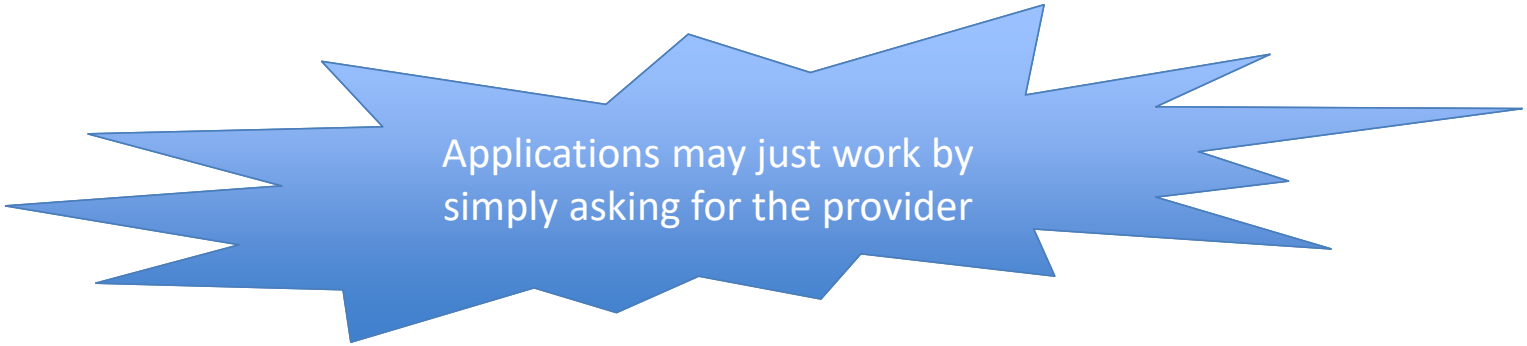
Working with Libfabric over Ultra Ethernet

LIBFABRIC UET PROVIDER SOFTWARE ARCHITECTURE

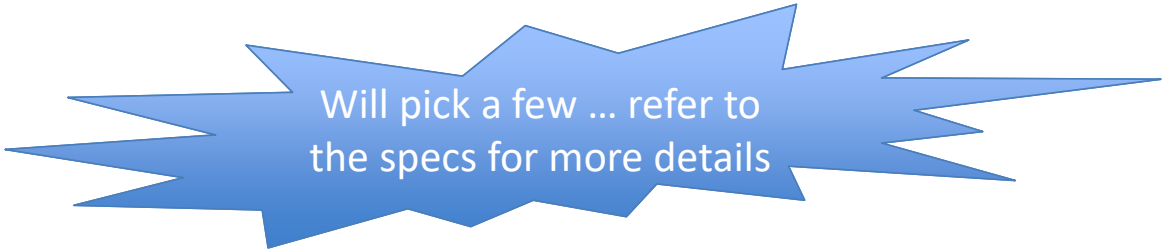


TO MAKE IT SIMPLE

- **Libfabric UET provider follows the same provider API as other providers do**



- **However, the UET provider do have some details that may need special attention in order to fully utilize the fabric features and avoid discovery errors**



UET ADDRESS (ENDPOINT ADDRESS)

FI_ADDR_UET

```
struct uet_addr {
    uint8_t ver;
    uint8_t reserved;
    uint16_t flags;
    uint16_t fep_cap;
    uint16_t pid_on_fep;
    struct uet_fa fa;
    uint16_t start_resource_index;
    uint16_t num_resource_indices;
    uint32_t initiator_id;
};
```

32 bytes

0

- Fields below valid?
- Relative / absolute addressing?
- IPv4 / IPv6?
- msg_size limited to MTU?

Bits for AI Base / AI Full / HPC

16 bytes

```
struct uet_fa {
    union {
        uint32_t v4;
        uint8_t v6[16];
    }
};
```

UET PROVIDER: DISCOVERY

```
int fi_getinfo(uint32_t version,
               const char *node,
               const char *service,
               uint64_t flags,
               const struct fi_info *hints,
               struct fi_info **info);
```

must >= 2.0

non-NULL only when flags has FI_SOURCE

predefined service name (optional)

FI_SOURCE if node is non-NULL

Field	Value
addr_format	Must be FI_ADDR_UET
src_addr	NULL or a UET address
dst_addr	Must be NULL

Fields	Values
src_addr	Partial address is returned: version / flags/ type / FA / start RI / FEP capabilities (red on previous slide)
dst_addr	Will be set to NULL.

ENDPOINT ADDRESS ASSIGNMENT

- **Local fabric address is available in info->src_addr returned from fi_getinfo()**
- **Full endpoint address is assigned when the endpoint is created**
  - A privileged entity is in charge of assigning endpoint address
  - The privileged entity may reside in user space (e.g. as part of a provision system), the UET provider would talk to the kernel driver, which will relay the request to the privileged entity
  - The kernel driver may need to program the NIC with the info from the response (security, authorization)
  - The kernel driver relay the address info back to the UET provider
- **Info contained in address assignment request**
  - IP address, service
- **Info contained in the request relayed by the kernel driver**
  - IP address, service, pid
- **Info contained in the response to the kernel driver**
  - Full UET address, Job ID, security binding
- **Info contained in the response relay back to the UET provider**
  - Full UET address, Job ID
- **Full endpoint address is retrieved with fi_getname() call**

MEMORY REGION

- **UET memory regions are always associated with endpoints**
  - mr_mode **FI_MR_ENDPOINT** is required (set via hints->domain_attr->mr_mode)
  - After an MR is created with fi_mr_reg*(), extra steps are needed before it can be used:

```
fi_mr_bind(mr, ep_fid, 0);
fi_mr_enable(mr);
```

- **Memory keys have standard format (for both user supplied & provider chosen)**

  - **I**: Idempotent Safe. May be used as target for idempotent operations (operations can be applied more then once)
  - **O**: Optimized. Support optimized non-matching headers (small headers having less bits for rkey)

JOB ID & AUTHORIZATION

- Job ID identifies a group of processes that are allowed to communicate with each other, i.e., belonging to the same "job".
- Job ID is carried in the SES header and is used to authorize access to target buffer
- The assignment of Job ID can be complicated, but from user's point of view, it's either passed in by the user or set by the provider
- In libfabric API, Job ID is passed in as "auth_key", part of domain_attr, ep_attr, mr_attr.
- If auth_key is not set by the user, the provider would use the control API to get the Job ID from the privileged entity, or use a fallback Job ID if such mechanism doesn't exist
- Most applications deal with a single Job ID:

	User set	Provider set
domain_attr (default)	auth_key_size = 3, auth_key = Job ID	auth_key_size = 0, auth_key = NULL
ep_attr (if not default)	auth_key_size = 3, auth_key = Job ID	auth_key_size = 0, auth_key = NULL
mr_attr (if not default)	auth_key_size = 3, auth_key = Job ID	auth_key_size = 0, auth_key = NULL

AUTHORIZATION WITH MULTIPLE JOB IDS

- **Previous mechanism can handle EPs, MRs with different Job IDs but each only handles one**
- **Some applications need to have one EP handle more than one Job ID**
- **FI_AV_AUTH_KEY: allow author key be inserted into AV (to be used as "address")**
  - fi_av_insert_auth_key: get an address representing all endpoints with the same auth_key
  - fi_av_insert with FI_AUTH_KEY flag: get an address representing a specific endpoint with a specific auth_key
- **Domain and endpoint must be opened with proper support**

	User set, with FI_AV_AUTH_KEY
domain_attr	auth_key_size = FI_AV_AUTH_KEY, auth_key = NULL
ep_attr	auth_key_size = 0, auth_key = NULL
mr_attr	auth_key_size = sizeof(struct fi_mr_auth_key), auth_key = pointer to struct fi_mr_auth_key

```
struct fi_mr_auth_key {
    struct fid_av *av;
    fi_addr_t src_addr;
};
```

- **Receive buffer posted with FI_DIRECTED_RECV enabled would only match sends with the same auth_key as the src_addr**

DEAL WITH PROFILE LIMITATIONS

- **Some profiles, especially AI Base, has limited capabilities**
- **Most of the limitations can be discovered via various attributes in the 'fi_info' structure**
- **One specific limitation need to be discovered differently**
  - **The AI Base profile allows limiting send/recv to one MTU size, while supporting much larger RMA sizes**
  - **info->ep_attr->max_msg_size is the maximum size among all ops**
  - **To get/set size limit for individual category, need to use the fi_getopt() / fi_setopt() API**

```
int fi_getopt(struct fid *ep, int level, int optname, void *optval, size_t *optlen);
int fi_setopt(struct fid *ep, int level, int optname, const void *optval, size_t optlen);
```

  - **The related option names are:**

```
FI_OPT_MAX_MSG_SIZE
FI_OPT_MAX_TAGGED_SIZE
FI_OPT_MAX_RMA_SIZE
FI_OPT_MAX_ATOMIC_SIZE
```

```
FI_OPT_MAX_INJECT_MSG_SIZE
FI_OPT_MAX_INJECT_TAGGED_SIZE
FI_OPT_MAX_INJECT_RMA_SIZE
FI_OPT_MAX_INJECT_ATOMIC_SIZE
```

OPERATION MAPPING

Libfabric API	FI_EP_RDM	FI_EP_DGRAM
fi_send	UET_SEND	UET_DATAGRAM_SEND
fi_sendv / fi_sendmsg / fi_inject / fi_senddata / fi_injectdata	UET_SEND UET_DEFERRABLE_SEND# UET_RENDEZVOUS_SEND#	UET_DATAGRAM_SEND
fi_tsend* / fi_tinject*	UET_TAGGED_SEND UET_TAGGED_DEFERRABLE_SEND# UET_TAGGED_RENDEZVOUS_SEND#	N/A
fi_read*	UET_READ	N/A
fi_write* / fi_inject_write*	UET_WRITE	N/A
fi_atomic / fi_atomicv / fi_injectatomic	UET_ATOMIC	N/A
fi_atomicmsg	UET_ATMIC, UET_TSEND_ATOMIC	N/A
fi_fetch_atomic* / fi_compare_atomic*	UET_FETCHING_ATOMIC	N/A

depending on message size and profile in use

PACKET DELIVERY MODES

- **Four modes defined in Packet Delivery Sublayer (PDS)**
  - ROD: Reliable Ordered Delivery, deliver once and only once
  - RUD: Reliable Unordered Delivery, deliver once and only once
  - RUDI: Reliable Unordered Delivery of Idempotent Operations, may deliver multiple times
  - UUD: Unreliable Unordered Delivery, best effort
- **Delivery mode is part of PDS header**
  - The UET provider choose the mode based on ordering requirement of operations (invisible to user)
  - Mix of ROD and RUD/RUDI enables ordering message with unordered data → better performance
- **Mapping guidelines for delivery modes**

Endpoint type	Profile	Delivery modes
FI_EP_DGRAM	-	UUD
FI_EP_RDM	AI Base, AI Full	ROD, RUD
FI_EP_RDM	HPC	ROD, RUD, RUDI

UET PROVIDER RUNTIME PARAMETERS

Name	Function
UET_PROVIDER_SERVICE_PATH	Path to optional service config file
UET_PROVIDER_MSG_RENDEZVOUS_SIZE	Minimal message size to use rendezvous send
UET_PROVIDER_TAGGED_RENDEZVOUS_SIZE	Minimal tagged message size to use rendezvous send
UET_PROVIDER_MAX_EAGER_SIZE	Maximum amount of data to send with the initial rendezvous request
UET_PROVIDER_DEF_DATA_DC	Optional override of default DSCP codepoint for data traffic class
UET_PROVIDER_FALLBACK_JOBID_SUPPORT	Use a fallback Job ID if one cannot be obtained from the job provisioning system
UET_PROVIDER_INITIATOR_ID	Initiator ID for endpoints configured through the fallback Job ID mechanism

UPSTREAM INTEGRATION

- **Status as of today**
  - No UET provider exists in upstream libfabric repo
  - There is a reference provider implementation in UEC member private repo
  - Vendors are working on their own UET providers
- **Getting providers into upstream is highly encouraged!**
- **Libfabric core has many utility code that can be reused:**
  - HMEM support, dmabuf support
  - MR cache, memory monitor
  - Utility objects on which providers can build their objects: fabric, domain, endpoint, CQ, EQ, AV, etc
- **Value add with utility providers**
  - Hooking providers: profiling, tracing, performance statistics
  - Lnx -- combine multiple providers
    - shm + network provider: cover or improve scale-up path
    - multiple network provider / provider instance: multirail
    - Requirement: support FI_PEER

FOR MORE INFORMATION

- **UEC Specification v1.0**
  - https://ultraethernet.org/wp-content/uploads/sites/20/2025/06/UE-Specification-6.11.25.pdf
- **Libfabric man pages**
  - Current head: https://ofiwg.github.io/libfabric/main/man/
  - Current head, all-in-one: https://ofiwg.github.io/libfabric/main/man/onepage.html
  - v2.0.0: https://ofiwg.github.io/libfabric/v2.0.0/man/
- **Libfabric source code:**
  - https://github.com/ofiwg/libfabric
- **UET libfabric reference provider (for UEC member only):**
  - https://github.com/ultraethernet/uet-libfabric
- **OpenFabrics Alliance (OFA)**
  - https://www.openfabrics.org
- **Linux Foundation**
  - https://www.linuxfoundation.org

2025 OFA Webinar Series

THANK YOU

Jianxin Xiong

Intel Corporation