IOWN GLOBAL FORUM ™

Data-Centric Infrastructure Functional Architecture

Classification: APPROVED REFERENCE DOCUMENT

Confidentiality: PUBLIC

Version 2

March 2023

[DCI]

Legal

THIS DOCUMENT HAS BEEN DESIGNATED BY THE INNOVATIVE OPTICAL AND WIRELESS NETWORK GLOBAL FORUM, INC. (“IOWN GLOBAL FORUM”) AS A **APPROVED** REFERENCE DOCUMENT AS SUCH TERM IS USED IN THE IOWN GLOBAL FORUM INTELLECTUAL PROPERTY RIGHTS POLICY (THIS “REFERENCE DOCUMENT”).

THIS REFERENCE DOCUMENT IS PROVIDED ”AS IS” WITH NO WARRANTIES WHATSOEVER, WHETHER EXPRESS, IMPLIED, STATUTORY, OR OTHERWISE, INCLUDING WITHOUT LIMITATION ANY WARRANTY OF MERCHANTABILITY, NONINFRINGEMENT OF THIRD PARTY RIGHTS, TITLE, VALIDITY OF RIGHTS IN, FITNESS FOR ANY PARTICULAR PURPOSE, OR ANY WARRANTY OTHERWISE ARISING OUT OF ANY PROPOSAL, REFERENCE DOCUMENT, SAMPLE, OR LAW. WITHOUT LIMITATION, IOWN GLOBAL FORUM DISCLAIMS ALL LIABILITY, INCLUDING WITHOUT LIMITATION LIABILITY FOR INFRINGEMENT OF ANY PROPRIETARY RIGHTS AND PRODUCTS LIABILITY, RELATING TO USE OF THE INFORMATION IN THIS REFERENCE DOCUMENT AND TO ANY USE OF THIS REFERENCE DOCUMENT IN CONNECTION WITH THE DEVELOPMENT OF ANY PRODUCT OR SERVICE, AND IOWN GLOBAL FORUM DISCLAIMS ALL LIABILITY FOR COST OF PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES, LOST PROFITS, LOSS OF USE, LOSS OF DATA OR ANY INCIDENTAL, CONSEQUENTIAL, DIRECT, INDIRECT, PUNITIVE, EXEMPLARY, OR SPECIAL DAMAGES, WHETHER UNDER CONTRACT, TORT, WARRANTY OR OTHERWISE, ARISING IN ANY WAY OUT OF USE OR RELIANCE UPON THIS REFERENCE DOCUMENT OR ANY INFORMATION HEREIN.

EXCEPT AS EXPRESSLY SET FORTH IN THE PARAGRAPH DIRECTLY BELOW, NO LICENSE IS GRANTED HEREIN, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS OF THE IOWN GLOBAL FORUM, ANY IOWN GLOBAL FORUM MEMBER OR ANY AFFILIATE OF ANY IOWN GLOBAL FORUM MEMBER. EXCEPT AS EXPRESSLY SET FORTH IN THE PARAGRAPH DIRECTLY BELOW, ALL RIGHTS IN THIS REFERENCE DOCUMENT ARE RESERVED.

A limited, non-exclusive, non-transferable, non-assignable, non-sublicensable license is hereby granted by IOWN Global Forum to you to copy, reproduce, and use this Reference Document for internal use only. You must retain this page and all proprietary rights notices in all copies you make of this Reference Document under this license grant.

THIS DOCUMENT IS AN APPROVED REFERENCE DOCUMENT AND IS SUBJECT TO THE REFERENCE DOCUMENT LICENSING COMMITMENTS OF THE MEMBERS OF THE IOWN GLOBAL FORUM PURSUANT TO THE IOWN GLOBAL FORUM INTELLECTUAL PROPERTY RIGHTS POLICY. A COPY OF THE IOWN GLOBAL FORUM INTELLECTUAL PROPERTY RIGHTS POLICY CAN BE OBTAINED BY COMPLETING THE FORM AT: www.iowngf.org/join-forum. USE OF THIS REFERENCE DOCUMENT IS SUBJECT TO THE LIMITED INTERNAL-USE ONLY LICENSE GRANTED ABOVE. IF YOU WOULD LIKE TO REQUEST A COPYRIGHT LICENSE THAT IS DIFFERENT FROM THE ONE GRANTED ABOVE (SUCH AS, BUT NOT LIMITED TO, A LICENSE TO TRANSLATE THIS REFERENCE DOCUMENT INTO ANOTHER LANGUAGE), PLEASE CONTACT US BY COMPLETING THE FORM AT: https://iowngf.org/contact-us/

Contents

List of Figures

List of Tables

1. Introduction

1.1. Overview of the IOWN Global Forum Architecture

Innovative Optical and Wireless Network Global Forum (IOWN GF) aims to establish an end-to-end architecture for computing and networking that can support various data flows and workloads, as shown in Figure 1.1-1.

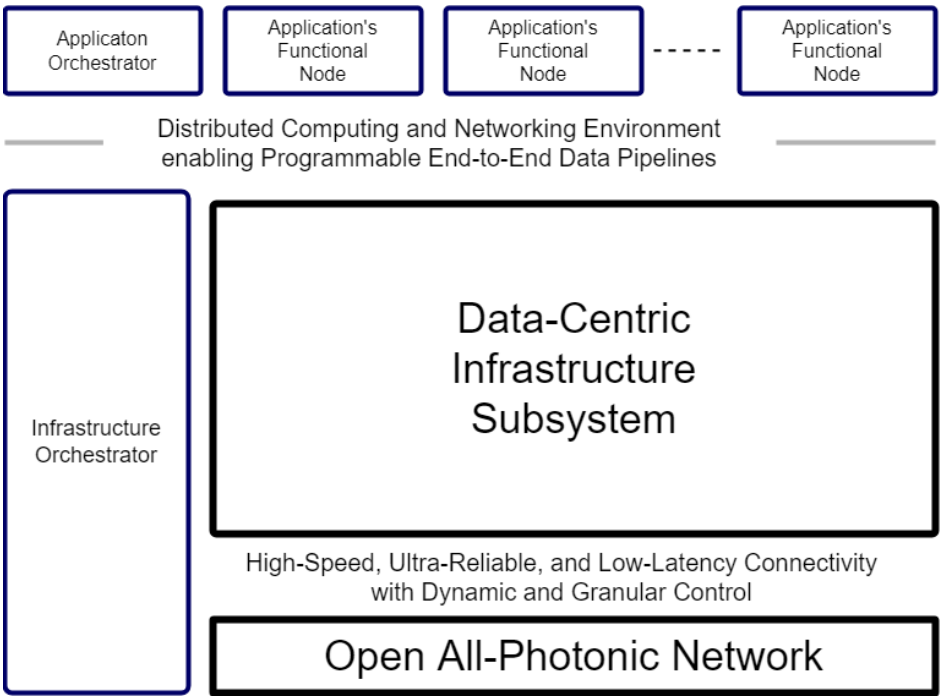


Figure 1.1-1: IOWN Global Forum Overall Architecture

The Open All-Photonic Network (APN) is a network that connects endpoints directly with optical paths. It provides high-speed, ultra-reliable, and low-latency connections. In today's networks, optical paths are disjointed and operated on a segment-by-segment basis, i.e., local area network (LAN), access network, and inter-data-center network. By contrast, the IOWN Global Forum Open APN will enable one optical path to span multiple segments. This will enable end-to-end communication with deterministic quality. However, this approach will require more dynamic and granular control for end-to-end optical path management.

Furthermore, as optical paths are dynamically created (making their performance demands impossible to predict until they are provisioned), we need a real-time performance measurement and monitoring mechanism that enables the infrastructure to set up new optical paths based on the projected achievable transmission speed. The IOWN GF aims to establish an open architecture for photonic networking so that service providers can integrate photonic network functions with their entire computing and networking infrastructure with more granularity. The open architecture should also enable service providers to build an intelligent operations support system.

The Data-Centric Infrastructure (DCI) subsystem is intended to provide applications with a distributed and heterogeneous computing and networking environment that spans end-to-end, i.e., across clouds, edges, and customer premises. This end-to-end, heterogeneous, and distributed computing/networking will enable service providers to build end-to-end data pipelines, placing data processing and storage functions in desired places. Data processing functions include filtering, aggregation, and event brokerage. Data storage functions provide shared storage, such as object storage and database, for data pipelines with multiple data sources and sinks.

DCI's support of heterogeneous networking will allow service providers to select data transfer and network protocols on a path-by-path basis. For example, protocols supporting deterministic quality may be used for network paths connecting real-time sensors in a manufacturing setting, while traditional IP networks would be used for networking paths connecting external data consumers. In this way, service providers will be able to accelerate data flow without isolating their systems from today's Internet ecosystems.

DCI's support of function-dedicated computing (FDC) will enable service providers to add various types of computing resources for performing dedicated computing tasks such as image artificial intelligence (AI) inference, time-sensitive data processing, network function virtualization (NFV), and database acceleration. In this way, service providers will benefit from the ongoing evolution of computing acceleration technologies.

The DCI subsystem exposes service interfaces to the Application's Functional Nodes for applications such as cyber-physical systems (CPS) and AI-integrated communication (AIC). Application developers can then build applications leveraging the functions and features provided by DCI and the Open APN. The features for high quality-of-service (QoS) are provided by the Function Dedicated Network (FDN) layer and may be realized by underlying networks, including an Open APN network.

The DCI Infrastructure Orchestrator (DCIIO) is the infrastructure's central management function that controls various infrastructure resources and exposes the single management interface. It is logically a single component, but it may be implemented with multiple nodes.

The Application Orchestrator is the central manager of an application system, which controls multiple application processes, i.e., microservices, for the application. When it deploys an application process on an IOWN Global Forum System, it should call the application programming interface (API) of the infrastructure orchestrator to create a runtime environment, e.g., a logical node.

Open APN, DCI, and Function Dedicated Network (FDN) are the main components of the IOWN GF System. Details of the Open APN are presented in the Open APN technical reports [IOWNGF-APN]. Details of DCI are presented in this DCI technical report.

1.2. FDN (Function Dedicated Network)

The FDN is a logical network created over DCI, Open APN, and Extra Networks and provides network connectivity with the quality required for IOWN GF services.

Note: The details of the FDN layer and its components will be further discussed and addressed in the future release. Only the overall concept is described in this document. Figure 1.2-1 shows an example of how FDN is configured.

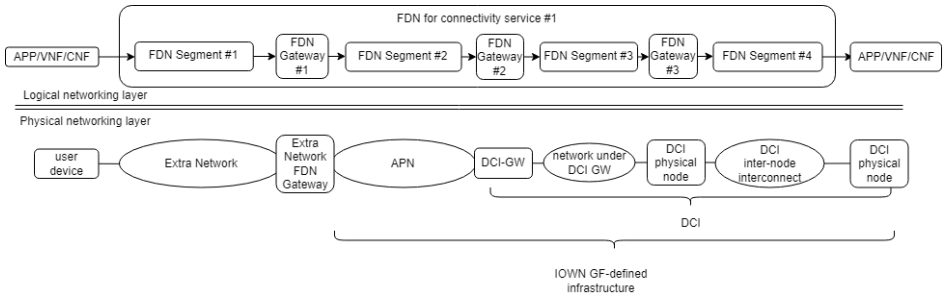


Figure 1.2-1: Example of FDN configuration

The following components are defined to illustrate FDN.

- **Function Dedicated Network (FDN)**: A logical network that provides connectivity between application data transport endpoints, designed to guarantee the QoS requirements of the application. FDN is created for specific classes/functions defined by supported network protocols and management/control APIs. FDN class

examples include Ethernet Virtual Connection (EVC), Converged Enhanced Ethernet (CEE), Time-Sensitive Network (TSN), Serial Digital Interface (SDI), Multi-Protocol Label Switching (MPLS), and the Internet Protocol (IP).

- **FDN Segment**: A logical subdivision of an FDN (e.g., an IP subnet, an L2 segment, and an L1 optical path). An FDN Segment is created within a single network infrastructure, for instance, an FDN Segment within Open APN or DCI intra-node interconnect. Multiple FDN Segments may be created within a single network infrastructure. FDN Segments may be interconnected by FDN Gateway functions that perform packet forwarding, protocol translations, etc., to form an FDN. However, the details of how an FDN Segment is created are left for the implementation.
- **Extra Network**: A physical network that is controlled, managed, and operated outside of the DCI cluster and the Open APN. An Extra Network connects with the Open APN through Extra Network FDN Gateway. Typical examples are local networks deployed in/near the customer premises and aggregate traffic from end nodes. FDN Segments can be instantiated on top of the Extra Network.
- **FDN Gateway**: A logical entity that connects one or more FDN segments with each other or their surroundings; FDN Gateway instances may provide functionality for the network infrastructure, such as routing and firewalling.

As the Open APN will provide optical paths that are more granular than those used for inter-data center communication, many end nodes, e.g., mobile radio units and user devices, will be connected directly to the Open APN. However, in some cases, users may choose to have an Extra Network between the Open APN and end nodes for several reasons. One reason may be because the Extra Network already exists and has been connected with end nodes. Another reason may be because the user wants to reduce cost of the optical transceiver by multiplexing the traffic of multiple nodes over one fiber. In such deployment scenarios, the Extra Network should support QoS-aware network protocols to avoid spoiling the benefits of optical communication.

In Figure 1.2-1, the FDN for connectivity service #1 is decomposed into several individual FDN Segments. These FDN Segments are interconnected by FDN Gateways that provide connectivity between different types of FDN Segments (in the figure, FDN Gateway #1 provides interconnectivity between FDN Segment #1 and FDN Segment #2).

1.3. Reference architecture

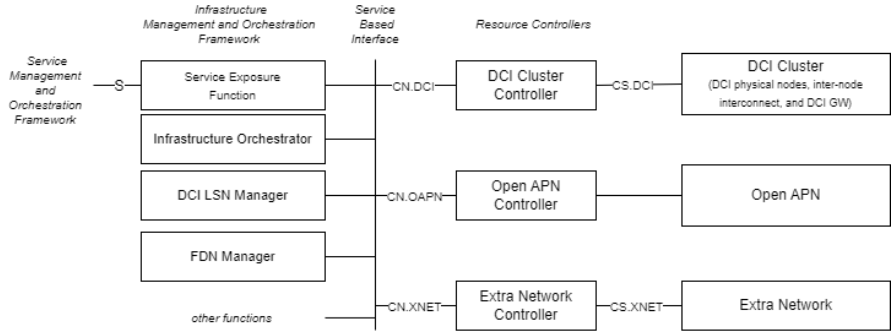


Figure 1.3-1: IOWN Global Forum Reference Architecture

Figure 1.3-1 shows the reference architecture defined by the IOWN Global Forum. This architecture is intended to realize elastic and scalable assignment of heterogeneous computing and networking resources to achieve data forwarding and processing under extreme QoS requirements.

The following functional blocks are defined as an initial reference. Other functional blocks may provide advanced features, but the details are deferred to IOWN GF service implementers.

- **DCI Cluster:** A computing infrastructure that comprises multiple DCI Physical Nodes/devices, an Inter-node Interconnect, and a DCI Gateway.

- **DCI Cluster Controller (DCICC)**: A logical function that dynamically reconfigures DCI Physical Nodes, the inter-node interconnect, and the DCI Gateway to create and maintain logical service nodes (LSNs) as defined below. The DCI Cluster Controller should also provide telemetry collection services. In addition, the DCI Cluster Controller manages and controls FDN Segments inside the cluster. One DCI Cluster Controller may control multiple DCI clusters.
- **Open APN Controller**: A logical function that controls the Open APN to create and maintain Open APN optical paths. The details should be defined in the Open APN Functional Architecture document.
- **Extra Network Controller**: A logical function that controls Extra Networks to create and maintain FDN Segments with enhanced elasticity, scalability, availability, and security. It should also provide a telemetry interface to collect the status data of FDN Segments. The supported set of APIs for management and control should be defined for each class of FDN.
- **DCI Logical Service Node (LSN)**: A logical node made of the physical computing and networking resources of a single DCI Cluster (see Appendix B.1 for further descriptions of LSNs and related concepts).
- **DCI LSN Manager**: A logical instance of a function that manages LSNs. The function includes installing software and configuring low-level parameters such as management network address and admin credentials. When a service provider has a service node manager for further configuration, the configuration by the DCI LSN Manager would be limited to the minimum installation and configuration necessary for the service provider's service node manager to access the LSN. While DCI LSN Manager may be implemented as a cluster of multiple nodes, it should expose a single point of the northbound interface and maintain single coherent storage of the managed data.
- **FDN Manager**: A logical instance of a function that creates and maintains instances of FDN with the desired QoS. FDN Manager calls resource controllers (i.e. DCI Cluster Controller, Open APN Controller, and Extra Network Controller) to create and manage FDN Segments and instances of FDN Gateways within each physical infrastructure.
- **DCI Infrastructure Orchestrator (DCIIO)**: A logical instance of a function that creates and maintains overall logical resources within the IOWN GF architecture, such as composites of LSNs interconnected with necessary network resources based on user requests. It calls the LSN Managers, Open APN Controllers, Extra Network Controllers, DCI Cluster Controllers, and FDN Managers to orchestrate logical resources.
- **Service-based interface**: A logical concept of connectivity among management and control plane functions. Functions interface with each other through service function calls.
- **Service Exposure Function**: A logical function that exposes IOWN GF system operation services to external users. Details of IOWN GF system operation services are deferred to DCI implementers.

This technical report provides more details of the DCI part of the IOWN GF reference architecture and the data plane acceleration.

2. Issues and gaps

The following issues are identified in today's communication and computing infrastructure.

- Scalability issue
    - Intra data center scalability issue: diverse compute workload imposes different computing, memory, and input/output (I/O) requirements. This requires that different components can be scaled differently. However, today's monolithic model is inflexible and inefficient in supporting flexible scaling of the computing, memory, and I/O resources.
    - Device, edge, and inter-data center scalability issue: Today's computing is centralized in the central cloud and regional edge clouds. Today’s centralized computing model is no longer sufficient for use cases with large data volume or high performance requirements. We expect a scalable solution to enable widely distributed computing across devices, far edges, regional edges and central cloud infrastructure.
- Performance issue:
    - Latency and jitter: some applications have stringent requirements on latency and jitter. Today's data transport schemes have high tail latency and jitter. There is a need to have data transport acceleration schemes to achieve controlled latency and jitter.
- Resource utilization and energy efficiency
    - Due to inefficiency in scalability, I/O bandwidth and memory size often becomes the bottleneck in resource provision, i.e., computing resources are often under-utilized due to the bottlenecks in I/O and memory. There is a need to research new schemes that can improve resource utilization and energy efficiency.

3. System architecture and services

3.1. Overview

Foreseeing the technology gaps described in Section 2, the data-centric infrastructure (DCI) provides the data, communication, and computing infrastructure for realizing the performance quantum leap and the advanced services that the IOWN GF envisioned. DCI aims to achieve the following design goals:

- **Inherent support for computing scaling out**: DCI shall have inherent support on scaling out computing across cloud center, cloud edge, network edge, and device
- **Heterogeneous Computing**: DCI shall support energy-efficient data-intensive computing that leverages a variety of computing accelerators
- **In-Network Computing**: DCI shall enable wire-speed data pipelines that receive and process data at the speed of the underlying transport network
- **Data Copy Reduction with Shared Memory/Storage**: DCI shall support computing with a shared storage/memory, enabling multiple data processing functions to access the same data in the shared storage/memory and thus reducing data transfer workload.
- **Heterogeneous Networking**: DCI shall support data flows of various QoS demands, including demands for reserved bandwidth, spike bandwidth for instant data transfer, very low latency, bound jitter, and time-sensitive networking.
- **Hub for Multiple Networks**: DCI shall enable data flows across multiple networks, e.g., IP and non-IP networks.

As shown in Figure 3.1-1, the DCI is built with computer clusters, called the DCI Cluster, that are interconnected with an Open APN. DCI Clusters form logical service nodes (LSNs), as defined in Section 3.2. For a workload requiring highly efficient data processing, LSN is enhanced with hardware accelerators that fit the type of the workload. The Open APN provides DCI clusters with optical paths for high-speed & ultra-low latency data transfer. DCI's infrastructure management & orchestration functions orchestrate the creation of LSNs and optical paths. In this way, DCI achieves streamlined data transferring and processing at the speed of optical communication.

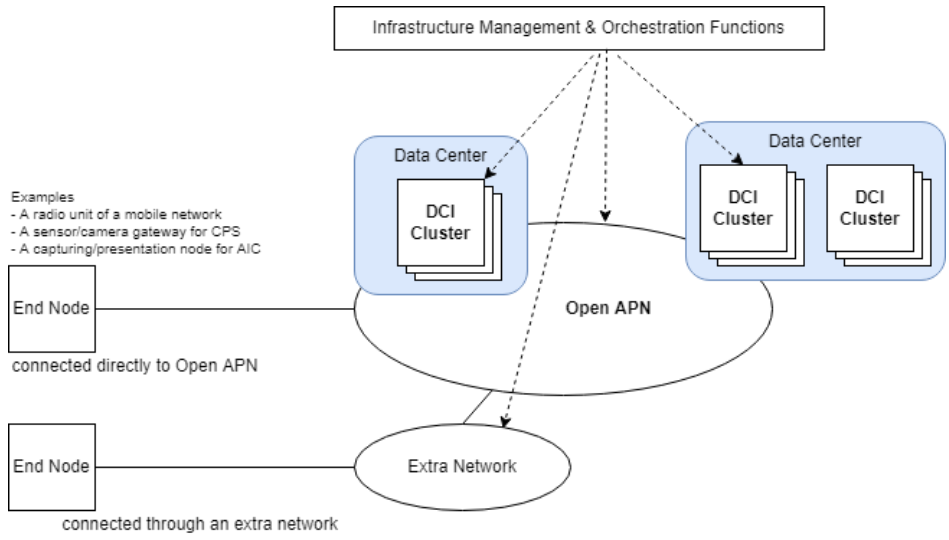


Figure 3.1-1: Overview of DCI Clusters and Open APN

DCI computing resource management is done in two layers:

- The DCI Cluster Controller should create and maintain logical service nodes (LSNs) at the hardware layer. Ideally, a cluster's physical elements, e.g., DCI Physical Nodes and interconnects, are only visible to DCI Cluster Controller and do not need to be exposed to external users. The DCI cluster controller can hide all resource details in the DCI cluster from external users. When needed, DCI Cluster Controller can announce information such as the total number of resources of each resource type and the number of free resources of each resource type. The Cluster Controller API would eventually support features like placement policy enforcement, as infrastructure users often demand these features.
- At the logical infrastructure layer, the DCI LSN Manager should configure LSNs. The logical node configuration by the LSN Manager means setting the parameters of logical resources of logical service nodes, e.g., hostname, network address, key certificate, software stack, Kubernetes node parameters, etc.

Note that the LSN Manager does not create, modify or delete LSNs. Instead, the DCI Cluster Controller creates, modifies and deletes LSNs, as this requires physical resource management.

3.2. DCI cluster

A DCI cluster is comprised of DCI physical computing nodes, Inter-node Interconnect, and DCI Gateway. An example of DCI cluster architecture is shown in Figure 3.2-1.

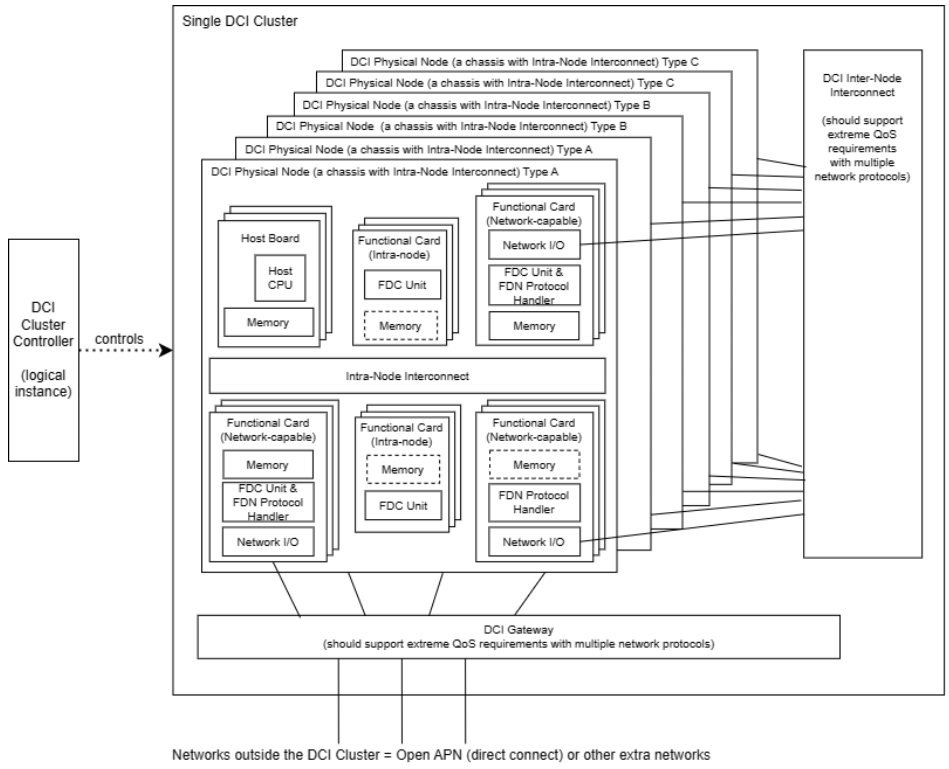


Figure 3.2-1: Example of DCI cluster architecture

3.2.1. DCI Physical Node, Intra-node Interconnect, and Functional Cards

A **DCI Physical Node** is a physical subsystem made of computing modules connected on an interconnect., which is referred to as an **Intra-node Interconnect**. Each DCI Physical Node should expose a management and control interface to the DCI Cluster Controller.

Computing modules include host boards and functional cards as defined below.

A host board is basically a physical computing module that has one or multiple host central processing units (CPUs) and memory.

A **Functional Card** is a physical module that performs some processing, be it data processing or network protocol unit processing, and is connectable to the intra-node Interconnect. Examples of the functional units are central processing units (CPUs), vector processing units (VPUs), graphics processing units (GPUs), reconfigurable processors, and network protocol handlers.

Traditionally, computing modules, e.g., CPU and GPU, and network interface modules, i.e., network interface cards (NICs), used to be separate. However, the demand for the efficient processing of large volumes of real-time data has led to the emergence of computing modules equipped with network interface modules. Smart NICs, Data Processing Units (DPUs), and Infrastructure Processing Units (IPUs) are examples of such functional cards. This enables us to offload large data copy traffic from the electronic-based internal interconnect to the optical-based inter-node interconnect. DCI is intended to take advantage of this trend. We express a functional card that has a network I/O as **Network-capable type**. By contrast, a card with no network I/O is expressed as **Intra-node type**.

An **Intra-Node Interconnect** is a module that connects one or multiple host boards with functional cards. The range of this Intra-Node Interconnect is confined within the range of the DCI Physical Node.

One choice of the intra-node interconnect is Peripheral Component Interconnect Express (PCIe). However, while PCIe has long been successfully adopted by many computing products, some of the features desired for disaggregated and heterogeneous computing are not provided. For example, it is not designed to accommodate multiple hosts. (This might be possible. But, at least, the multi-root configuration is not a common practice). In addition, PCIe does not natively support cache-coherency.

On the other hand, the computing industry has recently developed a new cache-coherent type of interconnect, which enables the connected modules to share one coherent memory space. One promising example of cache-coherent interconnect is Compute Express Link (CXL). In particular, CXL 2.0 supports the multi-host configuration. These features will be advantageous in building a heterogeneous and disaggregated computing infrastructure.

That said, IOWN GF DCI should support two choices for intra-node interconnect, PCIe, and CXL. Table 3.2-1 below compares PCIe and CXL.

Table 3.2-1: Comparison of PCIe and CXL

	DMA	MULTI-HOST	CACHE COHERENCY
PCIe	YES	Need to be checked (at least, not common)	NO
CXL	YES	YES (supported by CXL 2.0)	YES

Today's PCIe-based functional cards have local memory near functional units. But, a cache-coherent interconnect, such as CXL, enables functional cards to cache data from other modules directly. As a result, such functional cards may not have local memory. We expect this to reduce the total amount of memory in the computer, reducing energy consumption. Figure 3.2-2 illustrates various types of functional cards.

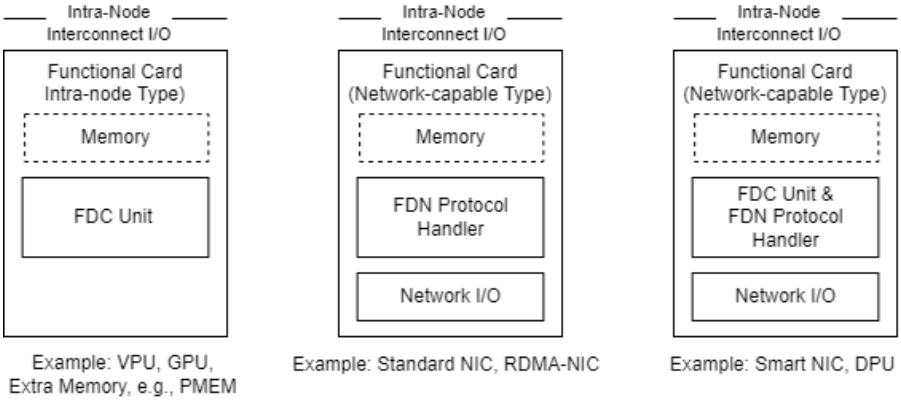


Figure 3.2-2: Types of functional cards

3.2.2. DCI Gateway

The **DCI Gateway** is a network module that relays data between DCI Physical Nodes in a DCI cluster and networks outside the DCI Cluster. It should provide data forwarding services that can achieve extreme QoS, e.g., guaranteed bandwidth of tens/hundreds of Gbps, forwarding latency less than a few microseconds for time-sensitive networking, required by IOWN GF use cases. It should also support point-to-multipoint forwarding.

It should expose a dynamic management/control interface to the DCI Cluster Controller to realize high availability and load balancing features.

A DCI Gateway as a logical instance should belong to a single DCI Cluster. For data transfer between DCI clusters, the DCI Gateways of the clusters should be inter-connected preferably with an Open APN. Furthermore, connectivity between multiple logical DCI Gateways could also be implemented within one physical DCI Gateway product that can be separated into multiple logical gateways.

IOWN GF will develop the reference implementation models (RIM) and specifications of the DCI Gateway and the Inter-node Interconnect during the technical specification phase. A short-term approach to implement a DCI Gateway and an Inter-node Interconnect would be to use a large-scale converged ethernet infrastructure. However, there is room for innovations for quantum leaps in capacity, low-latency support, scalability, and energy efficiency. In the long term, IOWN GF should drive these innovations leveraging optical communication and optoelectronics integration technologies. Therefore, the RIM will be defined with multiple classes.

3.2.3. Inter-node Interconnect

The **Inter-node Interconnect** is a network module that connects multiple DCI Physical Nodes to form a DC-scale computing infrastructure. The Inter-node Interconnect should provide data forwarding services that can achieve extreme QoS, e.g., guaranteed bandwidth of tens of Gbps, forwarding latency less than a few microseconds for time-sensitive networking, required by IOWN GF use cases. It should also support point-to-multipoint forwarding.

It should expose a dynamic management/control interface to the DCI Cluster Controller to realize high availability and load balancing features.

IOWN GF will develop the reference implementation models (RIM) and specifications of the DCI Gateway and the Inter-node Interconnect during the technical specification phase.

3.3. DCI Cluster Controller

The DCI Cluster Controller is a logical function that dynamically reconfigures DCI Physical Nodes, the inter-node interconnect, and the DCI Gateway to create and maintain **Logical Service Nodes (LSN)**. It exposes an API to DCI's

Infrastructure Management and Orchestration functions. In addition, as is common to DCI resource controllers, the DCI Cluster Controller should also provide telemetry collection services.

A DCI Cluster Controller may control multiple DCI Clusters.

4. DCI Infrastructure as a Service

4.1. Overview

DCI manages infrastructure resources dynamically to provide users with virtually private computing and networking infrastructures on an as-a-Service (aaS) model. The basic unit for networking service is an FDN Segment, and the basic unit for computing service is a logical service node (LSN).

This section defines DCI Infrastructure as a Service with its object model, APIs, and user personas.

4.2. Object Model

The figure below shows an object model for DCIaaS, which shows logical instances of provided infrastructure resources as objects. We call them DCI objects. DCI objects are associated with others, as shown in Figure 4.2-1.

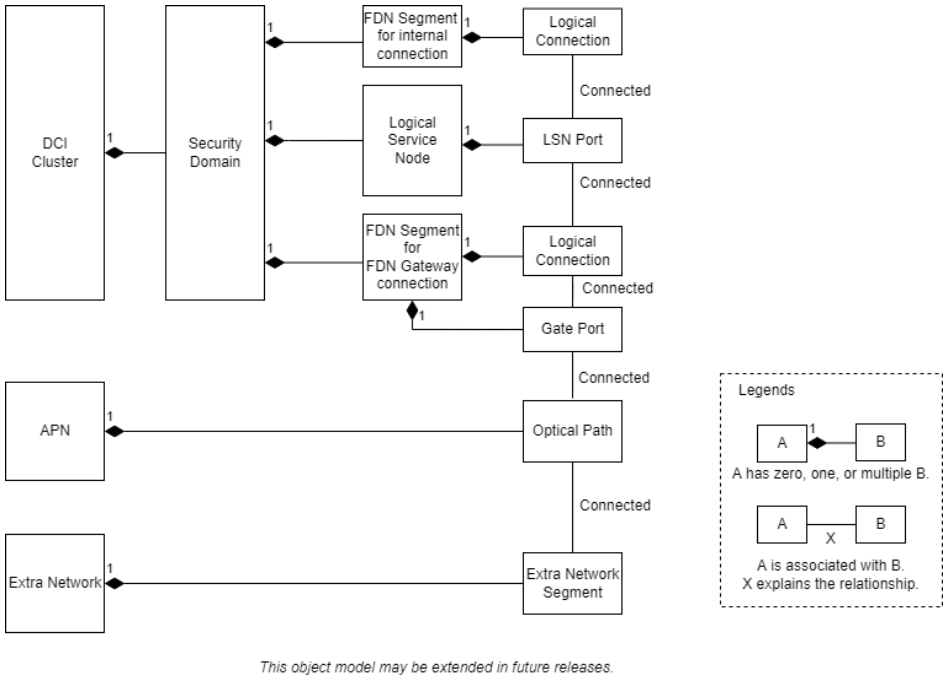


Figure 4.2-1: Object Model for DCIaaS

The definitions of DCI objects are given below.

- **DCI Cluster**: See Section 1
- **Security Domain:** A logical subspace of the computing space of a DCI Cluster, which a user can own as their private space. It is protected as an independent domain in access management. In other words, without explicit "allow" configurations on logical ports, access from external entities should be prohibited by default.
- **Logical Service Node (LSN)**: See Section 1.
- **LSN port**: logical port of LSN.
- **Gate port**: logical port of DCI Gateway.
- **FDN Segment**: See Section 1. Examples of FDN Segment would be a connection internal to a DCI Cluster, a connection that is created on an Extra Network, and other connections that bridge between DCI LSNs and networks outside the DCI Cluster.

- **Logical Connection**: A logical connection between/among FDN Gateways. It may be point-to-point or point-to-multipoint. When the FDN Segment is of a connectionless type, a set of forwarding configurations to enable data traffic between/among the connected logical ports is regarded as a logical connection. Logical connections achieve QoS-aware data forwarding.
- **Extra Network**: See Section 1.
- **Extra Network Segment**: An FDN Segment that is instantiated on top of an Extra Network.

Each DCI object has attributes. IOWN GF will clarify the attributes of DCI objects during the technical specification phase. However, some of the important attributes are noted in Table 4.2-1 below:

Table 4.2-1: DCI object attributes

DCI OBJECT	ATTRIBUTES (NOT EXHAUSTIVE)
DCI infrastructure owner	• user with usage time scale
Security Domain	• the owner
Logical Service Node	• the node flavor/type • the node recipe (See Section **Error! Reference source not found.**) • the resource allocation parameters • the telemetry parameters (Read-Only)
FDN Segment	• the FDN class/type • the telemetry parameters (Read-Only)
FDN Gateway	• the gateway type • the network protocol parameters • the telemetry parameters (Read-Only)
Logical Connection	• the connection/forwarding parameters • quality of transport service parameters (latency, data rate, packet drop rate, level of synchronization, availability, downtime, etc.) • the security policies • the telemetry parameters (Read-Only)
Optical Path	• the optical communication parameters • the MAC layer parameters • the telemetry parameters (Read-Only)

By setting the attributes of DCI objects, a DCI user can chain up DCI objects to form a DCI object composite (DCIOC) that can transfer and process data, accommodating extreme QoS requirements. Examples of DCI object composites are shown below in Figure 4.2-2.

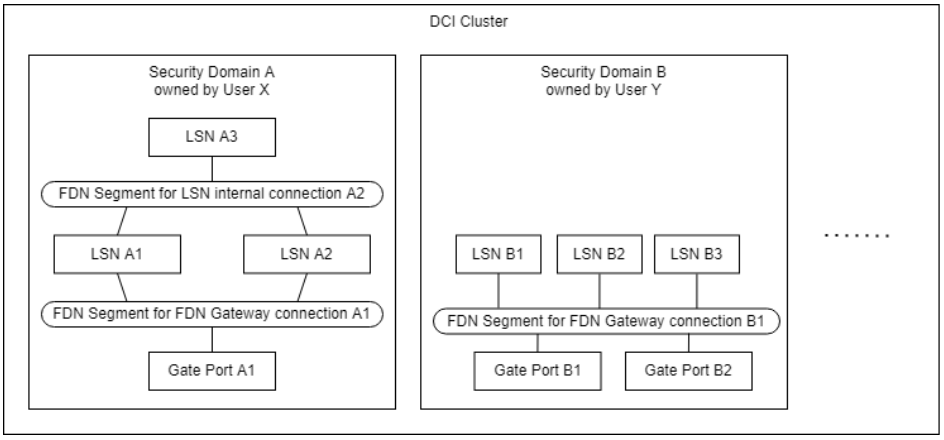


Figure 4.2-2: Examples of DCI object composite in a DCI cluster

4.3. Service APIs

Like today's cloud computing services, DCIaaS enables users to create/delete DCI objects and set/get DCI object attributes. In other words, DCIaaS should enable the basic Create, Read, Update, and Delete (CRUD) operations of DCI objects.

While IOWN GF will define DCIaaS API during the technical specification phase, some of the important primitives are described below:

- DCI LSN creation. This service enables a DCI service consumer (e.g., tenant platform provider) to request the creation of DCI logical nodes. The DCI service consumer provides logical node specifications (e.g., required computing types/resources, required FDN type/capacity, etc.). Functions in the DCI infrastructure orchestration and management framework can then identify the necessary resources and create the DCI logical nodes.
- DCI LSN configuration. This service enables a DCI service consumer to update DCI LSN configurations, such as adding or releasing computing resources, adding or removing FDN resources, update computing resource settings, updating FDN settings, etc.
- DCI LSN release. This service enables a DCI service consumer to release DCI LSNs.
- DCI LSN reserved. This service enables a DCI service consumer to release to reserve an LSN.
- DCI LSN chaining and pipelining. This service enables a DCI service consumer to request the creation of a service pipeline with a chain of LSNs. Data flow can be flowed in the DCI infrastructure layer among the chain of LSNs according to the service pipeline with minimal upper layer software intervention. The chain of LSNs can belong to multiple DCI clusters across multiple physical sites (e.g., edge site, regional site, national site, etc.).
- DCI LSN status monitoring. This service enables a DCI service consumer to monitor and obtain LSN operation status such as ready for testing, ready for operation, in error/erred.

4.4. DCIaaS User Personas

4.4.1. Tenant Platform Service Providers

With its heterogeneous architecture, DCI will become an ideal infrastructure for many services such as mobile network service, CPS area management service, and AIC streaming service. These services are called **DCI Tenant Platform Services** because, in most cases, the infrastructures of these services are designed to be a platform that can accommodate various applications.

Providers of DCI Tenant Platform Services are referred to as **DCI Tenant Platform Service Providers**.

DCI Infrastructures as a Service should support critical requirements from DCI Tenant Platform Service Providers. For example, we should assume the following requirements:

- Requirement 1: As a tenant platform service provider, I would like to control, manage and operate a group of DCI logical service nodes by my service management and orchestration system to have **control over the service quality**.
- Requirement 2: As a tenant platform service provider or an enhanced LSN provider, I would like to enable some protective measures, such as storage encryption, on DCI logical service nodes to **protect my intellectual property** from leakage.

Requirement 1 implies that the consumers of DCIaaS APIs are service management and orchestration systems owned by tenant platform service providers. Figure 4.4-1 below shows the stack of systems for typical DCI use cases.

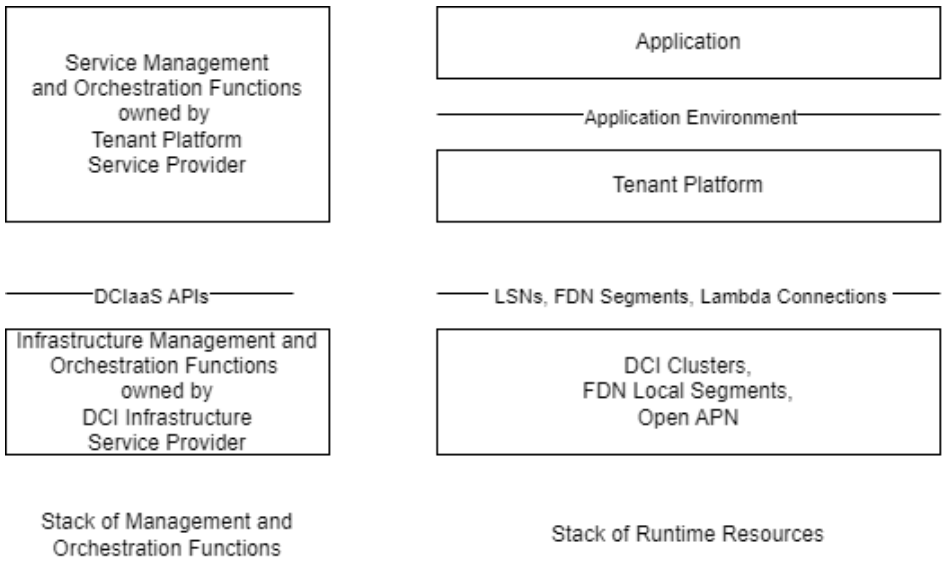


Figure 4.4-1: Tenant platform service provider stacks

Management and orchestration functions need to be selective to be provided under DCIaaS API to avoid **building layers of duplicative management and orchestration functions**. Therefore, the main functional scope of DCI's infrastructure management and orchestration functions should be the following:

- The resource allocation and configuration/re-configuration of physical infrastructure resources to create DCI objects defined in Section 4.2
- The configuration of the management port of the created DCI objects
- The installation and configuration/reconfiguration of the base software, e.g., OS and a minimum set of complementary software, for the created DCI objects
- The telemetry of physical infrastructure resources and DCI objects

For example, suppose a mobile network operator builds a virtual radio access network (vRAN) and cloud-native network function (CNF)/virtualized network function (VNF) platform on top of DCI as a tenant platform service provider. In this case, the mobile network operator's (MNO's) service management and orchestration (SMO) function would create LSNs invoking the DCIaaS API and add the created LSNs to its node list. IOWN GF will clarify the detailed procedure for this scenario during the technical specification phase. However, the following as a blueprint is assumed at this moment:

1. DCI LSN Manager allows tenant platform service providers to create their “node recipes,” which specifies the flavor of LSNs, installed base software, and initial parameter configurations.
2. MNO's SMO requests the creation of new LSNs, specifying the applied node recipe. This request would be sent to the DCI Infrastructure Orchestrator through the Service Exposure Function.
3. The DCI Infrastructure Orchestrator somehow resolves a DCI Cluster Controller and calls the cluster controller to create LSNs. Then, the DCI Cluster Controller creates LSNs, configures their management ports, and returns their management addresses.
4. The DCI Infrastructure Orchestrator calls the LSN Manager to install the base software into the created LSNs and apply the initial parameter configurations. This request should include the management addresses of the LSN and the ID of the applied node recipe.
5. Upon a completion response from the LSN Manager, the DCI Infrastructure Orchestrator sends to the MNO SMO a response including the LSN's management addresses.

4.4.2. Enhanced LSN Providers

While DCI Tenant Platform Service Providers would typically be regional businesses, they would seek the best-in-class technologies to handle a large amount of real-time data. This would create opportunities for product and equipment suppliers to develop logical service nodes (LSNs) specialized for specific purposes and sell them to the service providers in the global market. For example, a radio access network (RAN) vendor may allow its proprietary SDK and radio processing accelerator hardware to be added to LSNs and sell them to mobile service providers. We refer to these specialized LSNs as **enhanced LSNs** and refer to their suppliers as **enhanced LSN providers**.

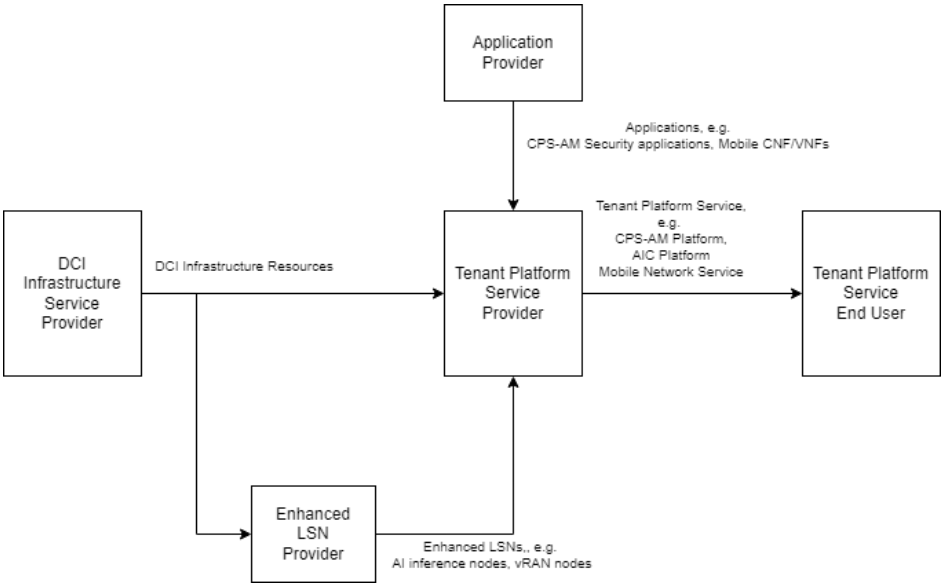


Figure 4.4-2: Relationships among the infrastructure service provider, tenant platform service providers, and enhanced LSN providers

Figure 4.4-2 illustrates the relationships among the infrastructure service provider, tenant platform service providers, and enhanced LSN providers.

IOWN GF will clarify the detailed procedures for the creation of enhanced LSNs during the technical specification case. However, we assume the following as a blueprint at this moment:

- An enhanced LSN provider registers its “node recipes” on the DCI LSN Manager.

- A tenant platform service provider selects the node recipe of an enhanced LSN provider in an LSN creation request to DCI.

5. Example deployment scenarios

5.1. Intra data center deployment scenario example

Figure 5.1-1 shows an intra-data center deployment example with PCIe for intra-node interconnect. Each DCI Physical Node comprises various functional cards that include computing devices (CPUs, GPUs, field-programmable gate arrays (FPGAs), or other types of accelerators), memory, NICs (including smart NICs and IPUs). The functional cards are interconnected via intra-node interconnect (such as PCIe, CXL, etc.). The DCI Physical Nodes in a DCI cluster are interconnected through inter-node interconnect. This Inter-Node Interconnect should be able to cater to extreme service requirements such as ultra-high throughput and ultra-low latency. A DCI cluster connects with networks outside the DCI Cluster via a DCI Gateway. Figure 5.1-2 shows an intra-data center deployment example with CXL for intra-node interconnect.

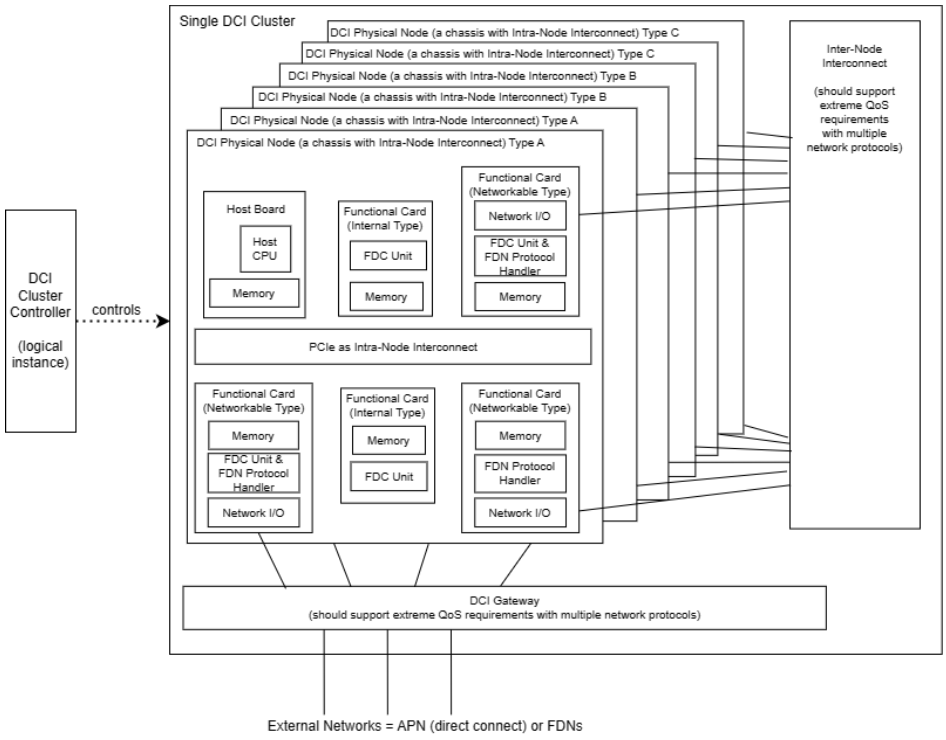


Figure 5.1-1: Intra data center deployment example with PCIe for intra-node interconnect

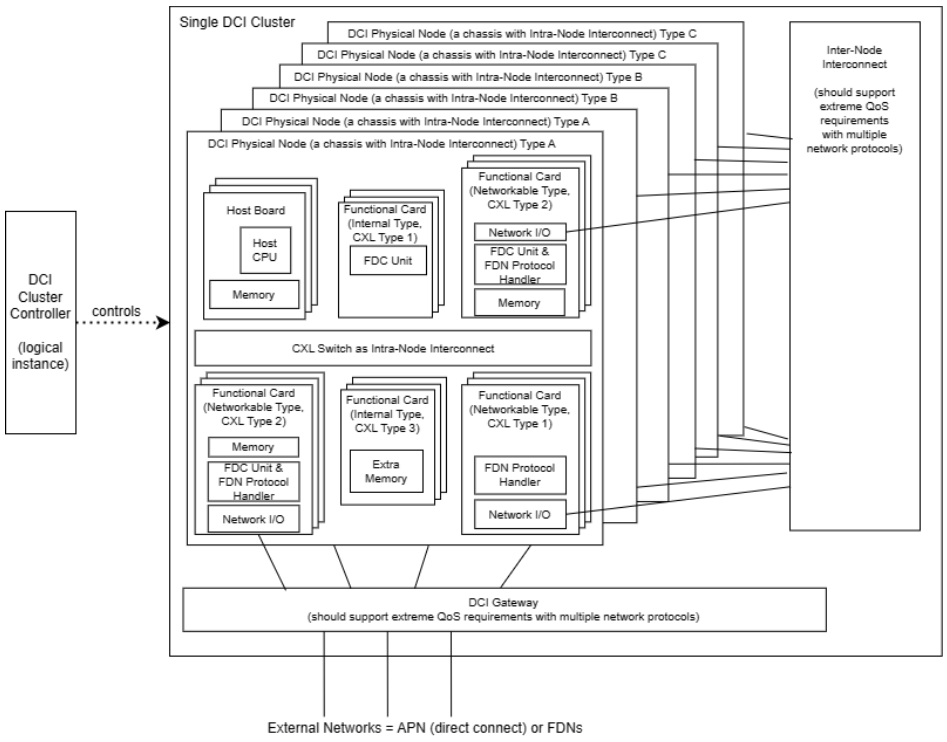


Figure 5.1-2: Intra data center deployment example with CXL for intra-node interconnect

Note that the difference between Figure 5.1-1 and Figure 5.1-2 is:

- PCIe will only allow one mainboard per DCI Physical Node.
- Some versions of CXL will allow multiple mainboards per DCI Physical Node; the actual transfer mechanism to be used is considered an item for further study.

5.2. CPS Area Management Security example

Figure 5.2-1 depicts an example of CPS Area Management with IOWN GF Open APN and DCI.

CPS Area Management builds data pipelines consisting of two levels of data aggregation and processing nodes called a local aggregation (LA) node and an ingestion (IN) node. The role of Infrastructure Management is to create DCI logical service nodes to host LA and IN and provision necessary connections. By contrast, the role of Service Management is to manage these LA and IN according to service demands. For example, a new demand may require adding a data processing function to some LA or IN. In this case, the service management system should deploy the data processing functions to the desired points.

We define Service Node Manager as a function of managing enhanced LSNs to host local aggregation (LA) and ingestion (IN) functions of the CPS area management (AM) use-case. The CPS-AM service provider may choose to build LA and IN as Linux container runtimes and deploy data processing functions in the form of Linux containers. In this case, the Service Node Manager would install the software stack for the Linux container runtime environment, e.g., Kubernetes software, into the DCI logical service nodes. Before this step, the DCI LSN Manager would only do minimal software installation and parameter configuration necessary for the CPS-AM node manager to access the logical service nodes.

We place an Extra Network to aggregate traffic from multiple LAs into an Open APN optical path. There would be many options for the data transfer protocol between LA and IN. If remote direct memory access (RDMA) is used, the Extra

Network should be capable of congestion avoidance and low latency. An example of such an FDN is Converged Enhanced Ethernet (CEE).

We assume that IN may be built with multiple LSNs in tiers because some use cases require one captured data to be analyzed by multiple AI inference applications.

A detailed CPS deployment model can be found in Section 5 of the Reference Implementation Model technical report [IOWNGF-RIM].

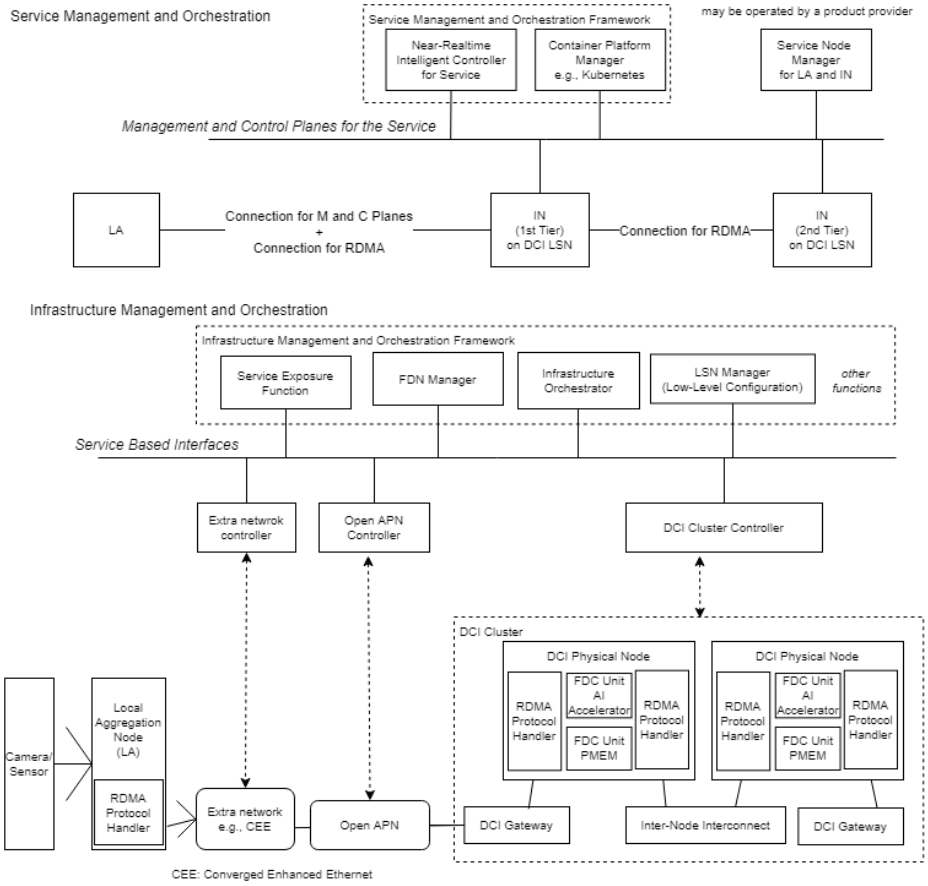


Figure 5.2-1: DCI for CPS Area Management Security

5.2.1. Operation Procedures

The deployment of a new IN should go through the following three stages:

- Stage 1 (Physical Resource Assignment)
  - DCI Cluster Controller creates DCI logical service nodes and connections inside the DCI Cluster.
- Stage 2 (Low-level Node Configuration)
  - DCI LSN Controller does minimum software installation and parameter configuration necessary for the Service Node Manager to access the created logical service nodes.
- Stage 3 (High-level Node Configuration)
  - The Service Node Manager does additional software installation and parameter configuration to make the created logical service nodes IN nodes.

The deployment of a new LA should go through the following two stages:

- Stage-1 (Physical Resource Assignment)
    - FDN Manager calls Open APN Controller, Extra Network Controller, and DCI Cluster Controller to create connections between the new LA and its IN:
        - DCI Cluster Controller creates connections between the DCI Gateway and the IN
        - Extra Network Controller reserves a port of the Extra Network for the new LA and creates connections between the reserved port and the Open APN port, i.e., the wide area network (WAN) port.
        - Open APN Controller creates or updates an Open APN optical path if necessary
- Stage-2 (Node Configuration through ZTP)
    - Once the new LA is connected to the port configured in Stage 1, some Zero-Touch Provisioning (ZTP) protocol should activate the new LA.

5.3. Mobile network deployment scenario example

Figure 5.3-1 depicts an example of disaggregated RAN with IOWN GF Open APN and DCI.

In this case, the role of Infrastructure Management is to create logical service nodes to host RAN functions and provisions necessary connections. By contrast, the role of Service Management is to deploy RAN functions to the created nodes and control the deployed RAN functions.

We define RAN nodes as enhanced LSNs to host radio unit (RU), distributed unit (DU), and central unit (CU) functions and RAN Node Manager as a function of managing RAN nodes, i.e., RU, DU, and CU. This management includes the high-level configuration of DCI logical service nodes, such as installing executable codes and setting RAN function parameters.

If a RAN product provider wants to provide its products via an as-a-service model without disclosing its intellectual property, we may choose to have the **RAN product provider operate the RAN node manager**. In this case, the DCI LSN Controller would only do minimal software installation and parameter configuration necessary for the RAN node manager to access the logical service nodes.

We place an FDN, called Mobile Fronthaul FDN, to aggregate multiple RUs into an Open APN optical path. This FDN should achieve strict mobile front haul transport requirements, such as latency and packet delay variation. An example of such an FDN is Ethernet Virtual Connection (EVC).

Detailed mobile network deployment model can be found in the Technical Outlook for Mobile Network Using IOWN Technology [IOWNGF-IMN].

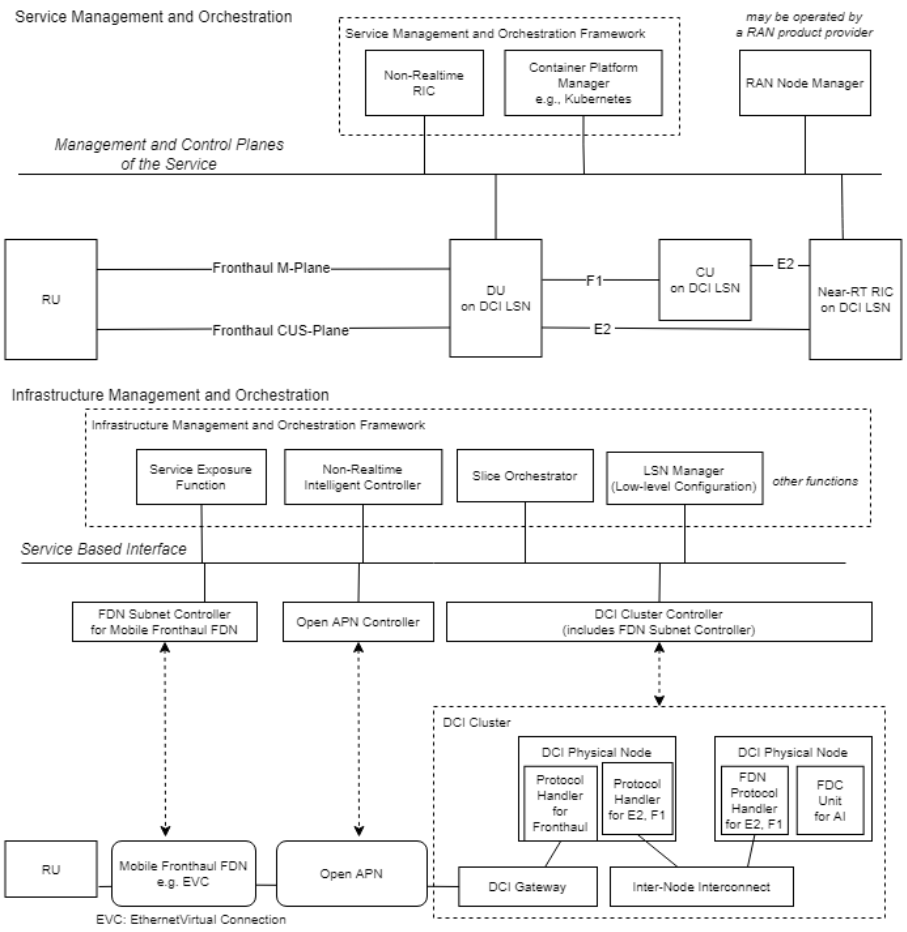


Figure 5.3-1: IOWN GF technologies for mobile network

5.3.1. Operation Procedures

The deployment of a new CU or DU should go through the following three stages:

- Stage 1 (Physical Resource Assignment)
  - DCI Cluster Controller creates DCI logical service nodes and connections inside the DCI Cluster.
- Stage 2 (Low-Level Node Configuration)
  - DCI LSN Controller does minimum software installation and parameter configuration that are necessary for the RAN Node Manager to access the created logical service nodes
- Stage 3 (High-Level Node Configuration)
  - The RAN Node Manager does additional software installation and parameter configuration.

The deployment of a new RU should go through the following two stages:

- Stage 1 (Physical Resource Assignment)
  - Infrastructure Orchestrator calls Open APN Controller, Extra Network Controller, and DCI Cluster Controller to create connections between the new RU and its DU:
    - DCI Cluster Controller creates connections between the DCI Gateway and the DU

        - Extra Network Controller reserves a port of the Extra Network for Mobile Fronthaul for the new RU and creates connections between the reserved port and the Open APN port, i.e., WAN.
        - Open APN Controller creates or updates an Open APN optical path if necessary
- Stage 2 (Node Configuration through ZTP)
    - Once the new RU is connected to the port configured in Stage 1, some Zero-Touch Provisioning (ZTP) protocol should activate the new RU.

The Okinawa Open Laboratory and NTT Group developed a proof concept system of the slice orchestration among RAN, Carrier Ethernet, and Optical Transport. Details are presented in [MEF3PoC].

6. DCI system control and management

This section provides high-level DCI system control and management procedures. These procedures could be updated in future releases.

6.1. Logical node management

6.1.1. Logical node creation

Figure 6.1-1 shows a procedure for LSN creation. In the procedure, the DCI service consumer requests LSN creation via the DCI service exposure function. The DCI service exposure function then forwards the creation request to the DCI infrastructure orchestrator. The DCI infrastructure orchestrator identifies a DCI cluster controller that can handle the request and sends the request to the selected DCI cluster controller. The DCI cluster controller then identifies the DCI cluster that can hold the LSN and sends an LSN creation request. Once the LSN is created, the completion message is sent back to the DCI service consumer. The DCI controller will also register the created LSN to the LSN manager. The completion message could contain the specifications of the created LSN, the LSN address information, and the LSN manager information.

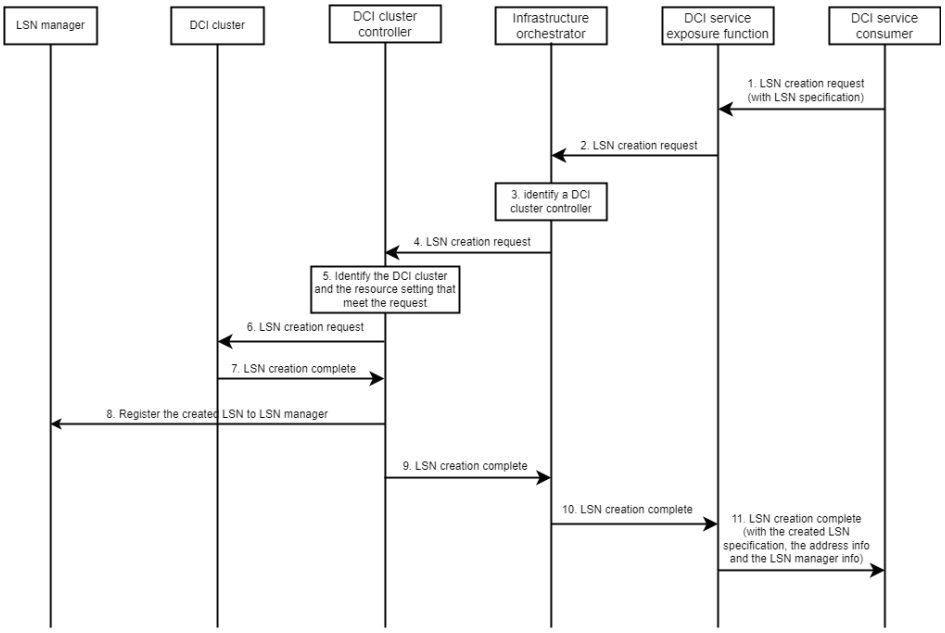


Figure 6.1-1: LSN creation procedure

6.1.2. LSN reconfiguration and update

Figure 6.1-2 shows a procedure for LSN configuration update. In the procedure, the DCI service consumer sends the LSN configuration request to the DCI service exposure function, which then forwards the request to the corresponding LSN manager. The LSN manger then updates the configurations of the LSN and updates the DCI cluster controller about the changes. Once the configuration is complete, the completion message is sent back to the DCI service consumer.

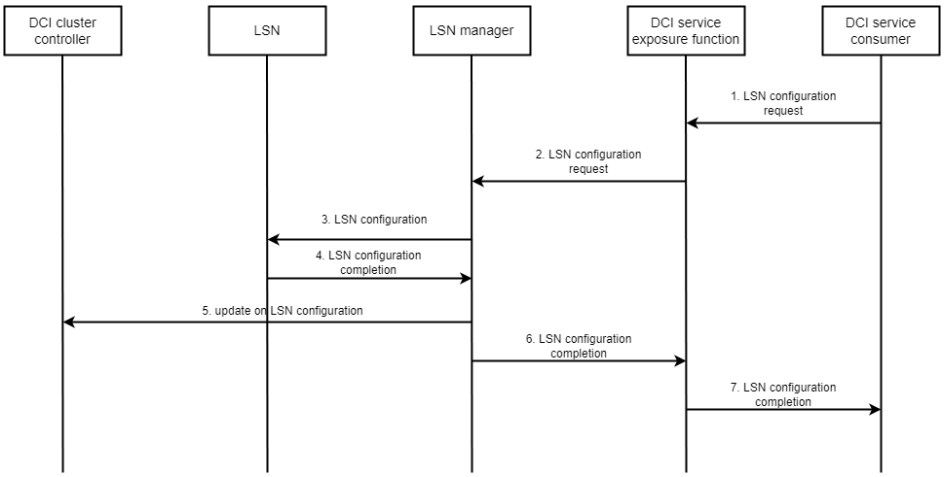


Figure 6.1-2: Procedure for LSN configuration update without physical resource addition/release in the LSN

The procedure shown in Figure 6.1-2 does not involve physical resource (compute/memory/FDN) addition/removal from the LSN. When there is a need to update the physical resource in an LSN, the DCI cluster controller needs to be called to handle the physical resource update. Figure 6.1-3 shows such a procedure. Steps 3-6 are the steps for physical resource update by the DCI cluster controller.

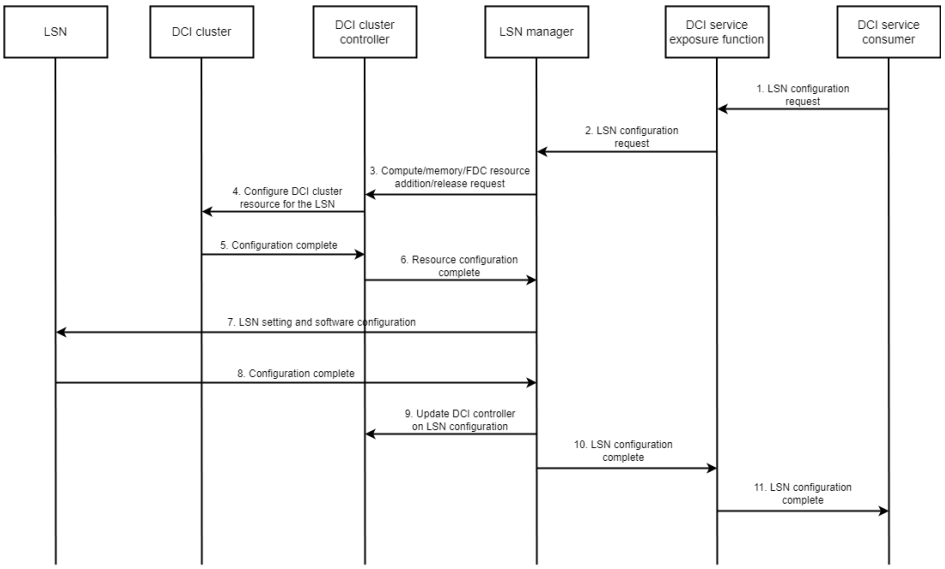


Figure 6.1-3: Procedure for LSN configuration update with physical resource addition/release in the LSN

6.1.3. LSN release

Figure 6.1-4 shows a procedure for LSN release. In the procedure, the DCI service consumer sends the LSN release request to the DCI service exposure function, which then forwards the LSN release request to the corresponding DCI cluster controller. The DCI cluster controller then deregisters the LSN from its LSN manager and releases the computing/memory and FDN resources assigned for the LSN. Once the LSN release is complete, the completion message is sent to the DCI service consumer.

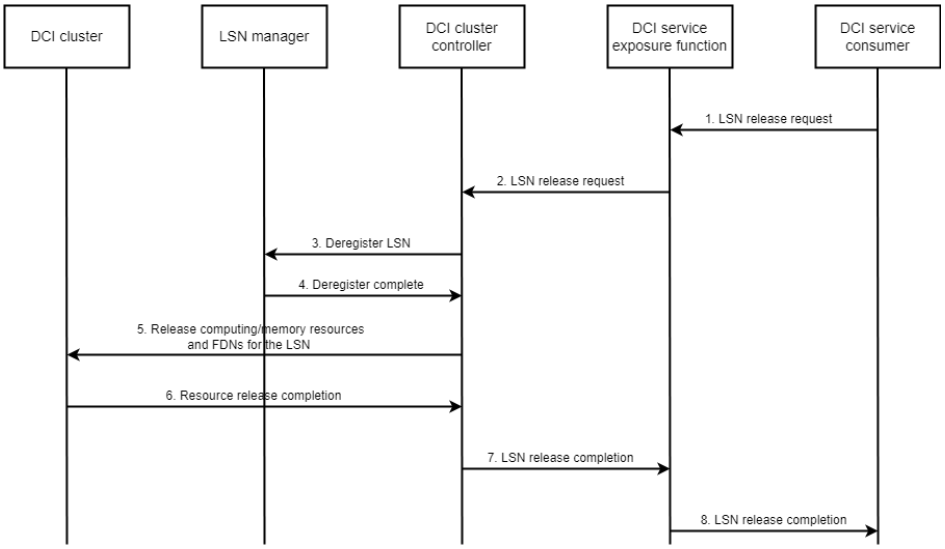


Figure 6.1-4: LSN release procedure

6.2. Hierarchical control and management

The DCI system's control and management hierarchy is comprised of the DCI infrastructure orchestrator, the DCI cluster controller, FDN/Open APN infrastructure controllers, and the LSN manager. The functionalities and responsibilities of those functions are described as follows:

- The DCI infrastructure orchestrator is the top-level control and management entity. It has the overall view of DCI resources and DCI controllers. Given an LSN create request, the infrastructure orchestrator will identify the DCI cluster that can hold the LSN and select the corresponding DCI cluster controller.
- The DCI cluster controller is responsible for the control and managing one or multiple DCI clusters. It has an overall view on the computing, memory, and FDN resources within those DCI clusters.
- The FDN/Open APN infrastructure controller is responsible for control and manages FDNs that interconnect DCI clusters.
- The LSN manager is responsible for boot up and configuring the LSNs. It does not directly manage the physical resources of the LSN. When there is a need to add or release some of the physical resources in the LSN, the DCI cluster controller will be called to conduct those tasks.

6.3. Orchestration and service chaining to generate the E2E data pipeline

Figure 6.3-1 shows a procedure for service orchestration and chaining. In the procedure, the DCI service consumer sends a service request to the DCI service exposure function, then parses the request and forwards it to the infrastructure orchestrator. The infrastructure orchestrator identifies and chains up the DCI clusters and the inter-cluster FDN for the service. The infrastructure orchestrator then sends corresponding configuration information to the DCI cluster controllers and the FDN controllers. The DCI cluster controllers and FDN controller then configure the DCI clusters and FDNs respectively to prepare the physical resources and connections for the service chain. Once the physical resource of the service chain is prepared, the infrastructure orchestrator can then send the service chain information to the LSN manger to prepare LSNs along the service chain. Once the LSNs along the service chain are configured, the service chain generation is completed. The DCI service consumer is then informed of the service chain completion.

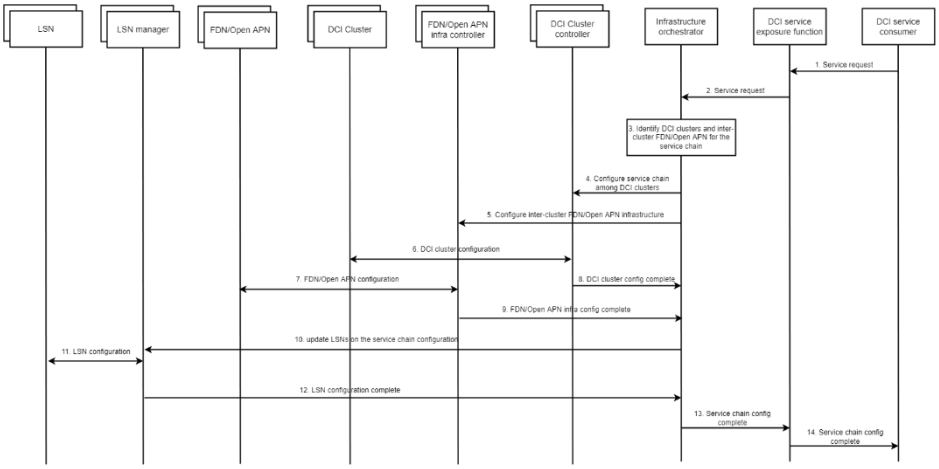


Figure 6.3-1: Procedure for service orchestration and chaining

7. DCI Cluster model

In previous sections, the DCI Cluster itself was described in Section 1.3, which puts the DCI Cluster into the context of the overall IOWN GF DCI Functional Architecture, Section 3.2, which illustrates one possible internal structure for DCI Cluster implementations, and Section 5.1, which gives examples of DCI Cluster implementations with concrete technologies. Furthermore, interactions of the DCI Cluster and its controller with other blocks of the IOWN GF DCI Functional Architecture are detailed in Section 6.

This section introduces an initial, basic, and formalized model of a DCI Cluster. This model describes how a DCI Cluster Controller (DCICC, cf. Figure 1.3-1) needs to present the DCI Cluster capabilities and current DCI Cluster status to management plane blocks of the IOWN GF DCI Functional Architecture such as the DCI Infrastructure Orchestrator (DCIIO, cf. Figure 1.3-1).

This model is not intended to specify or describe the actual realization of DCI Clusters. Implementors are free to choose any realization; however, DCI Cluster Controller implementations need to cause DCI Cluster implementations to behave according to the DCI Cluster model.

In the future, the DCI Cluster model might be extended, for example, to include additional DCI Cluster functionality or to make the DCIIO aware of further DCI Cluster implementation constraints. It is expected that considering such changes will regularly involve balancing the complexity of the DCI Cluster implementations against the complexity of infrastructure orchestration. Furthermore, later IGF technical reports might describe software APIs based on such a model.

The remainder of this section is structured as follows. First, the role of a formalized DCI Cluster model is outlined in Section 7.1. After that, the requirements for a DCI Cluster model are discussed in Section 7.2. Then, a basic DCI Cluster model is introduced in Section 7.3, which forms the core part of this section. Finally, directions for future work and possible extensions for the basic DCI Cluster model are proposed in Section 7.4. Furthermore, possible DCI Cluster implementation strategies are discussed in Appendix B.2, and an actual example of an instance of the basic DCI cluster model proposed in Section 7.3 is presented in Appendix B.3.

7.1. Role of a formalized DCI Cluster model

The IOWN GF use-case documents describe scenarios in which sets of (physical-)LSNs of IOWN GF applications are expected to be deployed in a geographically highly distributed manner. Single IOWN GF application deployments might even cross organization boundaries.

At the same time, different datacenters or organizations may employ multiple DCI Cluster hardware types in parallel. DCI Cluster hardware vendors may use different implementation strategies, and each single vendor may even offer different DCI Cluster implementations that optimize the structure of a DCI Cluster for specific use-cases. Therefore, to deploy applications, the DCI Infrastructure Orchestrator needs to be able to understand the state of all involved DCI Clusters, i.e., all DCI Clusters in which (physical-)LSNs for a given application are potentially created.

However, equipping an entity that is global to the whole IOWN GF system such as the DCIIO with detailed internal knowledge about each possible DCI Cluster seems to be infeasible. The reason is that such an entity would need intricate knowledge of all possible DCI Cluster implementations of all DCI Clusters in which an application could be deployed. (If the DCIIO had such wide-ranging and intricate knowledge of the internals of all DCI Clusters, the DCICC would not be strictly required as an independent block.)

Therefore, to resolve this issue, the role of the DCI Cluster Controller (DCICC) is to abstract the internals of DCI Clusters to the DCIIO, so that the DCIIO and the DCICC can be developed, maintained, and evolved independently.

More requirements toward such a model are discussed in the following section.

7.2. Requirements toward a simple DCI Cluster model

The previous section explained that the DCICC needs to provide a meaningful abstraction of the DCI Cluster controller state to the DCIIO. Therefore, there must exist an abstract model of DCI Cluster hardware on which both DCIIO and DCICC agree. As a starting point to construct such a model, the following section proposes a DCI Cluster model designed to be as simple as possible.

From the viewpoint of the DCI Infrastructure Orchestrator, the DCI Cluster Controller would ideally behave according to an abstract model as follows:

- The only limited resources visible to the DCIIO are the functional cards (CPU, GPU, IPU, etc.). The DCIIO only needs to be aware of the amount of computational resources and can query these resources. (A DCI Cluster model that provides unlimited computing resources is considered not to be useful since not implementable.)
- The DCI Cluster internally manages the interconnects inside the DCI Cluster automatically, so that the DCIIO does not need to be aware of the interconnects required for LSN formation. Additionally, all functional cards can be arbitrarily combined into (physical-)LSNs in any combination and size.
- The DCI inter-node interconnect always supports sufficient quality of service between all functional card ports.
- All incoming APN logical channels can be converted for and freely forwarded to any DCI inter-node interconnect port.
- There is a standardized format in which the boot images for some types of functional cards, and the DCICC has all required knowledge regarding how to use these boot images to autonomously perform lowest-level initialization of LSNs.

A simple DCI Cluster model according to the requirements above is presented in the following section.

7.3. Introduction of a simple DCI Cluster model

An idealized model of DCI Clusters based on the discussions above is shown as UML diagram in Figure 7.3-1. The model introduced here is intended to be simple to use by the *DCI Infrastructure Orchestrator*. The model is specifically not intended to be a guideline or blueprint for the implementation of *DCI Cluster* hardware.

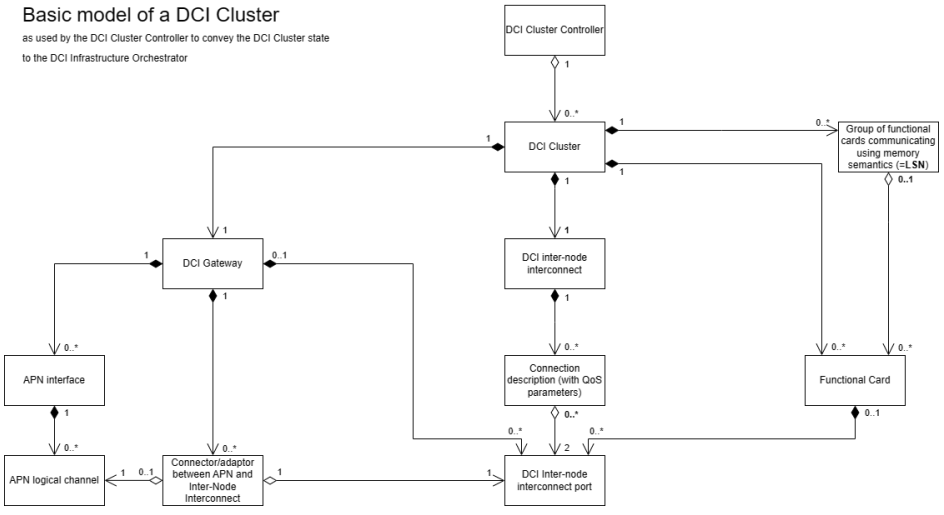


Figure 7.3-1: Basic model of a DCI Cluster

Elements of the model are introduced below:

- *DCI Cluster Controllers*. *DCI Cluster Controllers* control *DCI Clusters*. There is currently no restriction placed regarding how many *DCI Clusters* a single *DCI Cluster Controller* may control.
- *DCI Cluster*. *DCI Clusters* are the largest organizational unit of computing devices in *DCI*. In this model, *DCI Clusters* are composed of exactly one *DCI Inter-node Interconnect*, an arbitrary number of *Functional Cards*, a mechanism to compose an arbitrary number of *groups of functional cards communicating using memory semantics*, and exactly one *DCI Gateway*.
- *DCI Inter-node Interconnect*. The *DCI Inter-node interconnect* is used to connect *Functional Cards* to each other independent of the *LSNs* the *Functional Cards* belong to, and to connect *Functional Cards* to the outside of the *DCI Cluster* via the *DCI Gateway*. The connectivity the DCI Inter-node Interconnect provides is represented by *Connection Descriptions (with QoS parameters)*.
    - Currently, there is no concept of “inter-node” in this model besides communication between *Functional Cards* and *DCI Gateways*. This may change in future revisions.
- *Connection Description*. *Connection Descriptions (with QoS parameters)* are used to represent the connectivity between *DCI inter-node interconnect ports*. Every *Connection Description* aggregates exactly two *DCI inter-node interconnect ports* and defines the quality-of-service properties between these two ports.
- *DCI inter-node interconnect ports*. *DCI inter-node interconnect ports* model network ports on *Functional Cards*.
    - A given port may be referenced by any number of *Connection Descriptions* to realize arbitrary communication topologies.
- *Functional Card*. *Functional Cards* correspond to functionality required for computation, such as CPUs, GPUs, FPGAs, IPUs/DPUs, memory, storage, etc. (cf. Section 5.1). Each *Functional Card* may comprise an arbitrary number of *DCI inter-node interconnect ports*.
- *Group of functional cards communicating using memory semantics*. These groups of *Functional Cards correspond to logical service nodes (LSNs)* visible to the outside of the *DCI Cluster*.
    - Arbitrary Functional Cards may be combined into arbitrarily many groups of arbitrarily many Functional Cards.
    - A Functional Card may belong to no group/LSN (when the resource is not used) or to exactly one group/LSN. A Functional Card may not belong to more than one group/LSN.
    - Groups are allowed to contain no resources/Functional Card at all. The rationale is that since Functional Cards may be dynamically added and removed from LSNs, LSNs should be allowed to have no resources, so that LSN maintenance information, such as information regarding security domains and tenants to which the LSN belongs do not need to be set up repeatedly.
- *DCI Gateway*. *DCI Gateways* are responsible to handle the communication of *Functional Cards* with the outside of a *DCI Cluster*. *DCI Gateways* are composed of *APN Interfaces* and *Connectors/adaptors between APN and DCI Inter-node interconnect*.
- *APN interface*. *APN interfaces* correspond to physical interfaces to the APN, such as a single fiber core, which represents the smallest independent medium level. *APN interfaces* in turn are composed of *APN logical channels*.
    - It is expected that these *APN Interfaces* will be standardized throughout the whole IOWN GF Open APN network.
- *APN logical channels*. *APN logical channels* represent configurable transmission channels made available through an *APN interface*, such as wavelength channels, time-division multiplex channels, or combinations of such methods.
    - It is expected that these *APN logical channels* will be standardized throughout the whole IOWN GF Open APN network. Furthermore, such channels can be both digital as well as analog in nature.

Additionally, for digital channels, digital channel coding might be visible or opaque to the DCI Gateway. The exact options are for future discussions.

- *Connectors/adaptors between APN and DCI Inter-node interconnect*. These connectors receive and forward data or signals from the APN to *DCI inter-node interconnect ports* of *Functional Cards*.
    - It is expected that in the general case, the DCI Gateway will receive digital data coded for and transmitted by signals for long-range transmission (≫2km), and then forward this data coded for and transmitted by signals for short-range transmission within the *DCI Cluster*. A *DCI Gateway* may also offer the ability to directly forward the APN signals without recoding to the inside of the *DCI Cluster* to *Functional Cards* equipped with *DCI inter-node interconnect ports* (network interfaces) that are able to receive such APN signals.

Furthermore, the model introduced above does not contain blocks to represent *DCI Intra-node interconnects*, *Host boards*, or *Host CPUs*. *DCI Intra-node interconnects* and *host boards* are omitted, because infrastructure orchestration should ideally not have to be concerned with such implementation artifacts. In turn, the omission of such details from this model enhances the freedom for implementation for hardware providers, and the concrete implementation of the *DCI inter-node interconnect* will possibly become one key differentiator of actual products. *Host CPUs* are assumed to be treated in the same manner as individual *Functional Cards*.

An example of an instance of this model is presented in Appendix B.3 for illustration and to ease understanding.

The following section discusses possible future extensions of the DCI Cluster model.

7.4. Possible future extensions of the presented DCI Cluster model

A simple DCI Cluster model was introduced in Section 7.3. While it might already be possible to implement DCI Clusters and DCI Cluster Controllers according to this model, and further to specify and generate queries and commands from the DCI Infrastructure Orchestrator to the DCICC, actual products might require further extensions of this model to increase attainable performance and scalability, and further, foster interoperability discussions among hardware providers.

During the initial discussions about the DCI Cluster model as shown in Section 7.3, a number of points to consider for possible extension and/or refinement of the model were already proposed, which are listed below.

- DCI Inter-node interconnect details. With the proposed model, the DCI Inter-node interconnect QoS can only be configured between pairs of ports. Consideration is needed whether performing low-level configuration on a pair-to-pair basis is sufficient. Furthermore, more details about ports might need to be modeled, including for which network layer(s) properties can be specified.
- QoS. Currently, QoS parameters of Connection Descriptions as well as the QoS between resources/Functional Cards such as CPUs and memory are not explicitly included in the model. However, for actual operation such information is required to configure the QoS and functionality of DCI Inter-node interconnect between its ports and to steer resource selection, e.g., determine from how far apart locations devices and memories may be selected from according to user latency and bandwidth QoS specifications.
- FDNs. A key feature of the DCI architecture are function-defined-networks. Further consideration is needed if QoS objects of a DCI Cluster model are sufficient to create FDNs as introduced in Section 1.2 or whether further objects are required to set up FDNs by high-level management functionality outside of the DCI Cluster.
- Logically partitionable resources. In the current model, logically partitionable resources are not explicitly considered. For example, there is no specification regarding how the DCICC uses the model to convey available memory to the DCIIO. A simple approach to stay within the current model would be to announce a number of available 1GB memory Functional Cards to the outside that corresponds to the available memory

within the cluster. Whether extensions are necessary needs to be evaluated for a variety of logically partitionable devices such as memory, storage, SR-IOV devices, MIGs, and multi-core CPUs.

- Separated resource pools. Actual implementations of DCI Clusters might need to impose limitations on which Functional Cards can be combined into LSNs. It needs to be considered whether or not DCIIOs need to be aware of such restrictions, such as the existence of multiple separate resource pools or DCI Physical Nodes inside a single DCI Cluster.
- DCI Gateway details. This initial model provides a two-layered abstraction of physical APN interfaces and configurable channels within. However, this initial model does not model limitations regarding which APN channels can be forwarded to which ports. Whether this abstraction is sufficient needs to be further evaluated, likely together with the concerned task forces.
- DCI Intra-node interconnect. As also explained in Section 7.3, since this element is deemed to have large innovation potential and there is a possibility that the implementation of this interconnect will vary greatly among products, the DCI Intra-node interconnect is hidden from the outside of the DCI Cluster and the DCIIO. Communication via the DCI Intra-node interconnect can only be requested indirectly by the user via LSN creation in combination with QoS settings. However, there might be cases where disclosure of information including details of such interconnects might be permissible for debugging purposes, e.g., allowing administrators or users to query the location of LSN components and their connectivity.
- Host board. In the same manner as the DCI Intra-node interconnect, details about host boards should be hidden, but disclosure of limited information for debugging might be permissible.
- DCI Security domains are not explicitly modeled yet. Since the DCI Cluster Controller needs to take into account such security settings, consideration is required how to extend the DCI Cluster model to allow the DCIIO to set security-related properties of LSNs.
- Resource/Functional Card initialization. Currently, there has been no consideration yet whether the initialization of resources needs to be explicitly considered in the DCI Cluster model. For example, possibly Functional Cards could be given a type attribute that also determines how/if a given Functional Card can or needs to be initialized.

The above points will be further evaluated in the future regarding whether or how these points need to be reflected in the DCI Cluster model.

8. Data Plane Acceleration

Due to the increase of network bandwidth, applications running on current server systems consume large amounts of CPU processing power not only for application specific tasks, but also for network protocol processing. To simultaneously provide enough bandwidth for the extreme use cases of the IOWN Global Forum and reduce the (system) power consumption, it is essential to create a data plane acceleration framework that provides mechanisms including offloading the task of network processing into hardware accelerators.

8.1. Overview of IOWN Global Forum Data Plane Acceleration Framework

8.1.1. Introduction

The IOWN GF use cases require to support various types of data processing and data flow, implemented on different data planes. The existing data plane framework may be able to be applied to some data flows. A new data plane framework may be needed for other data flows. In this section, the data plane acceleration (DPA) framework aims to apply to such latter data flows in combination of different data plane types. For this purpose, first, we analyze data flows of the CPS AM use case described in the RIM document [IOWNGF-RIM] and IDH document [IOWNGF-IDH]. Then, we define data plane types in terms of communication range from internal communications within a DCI Physical Node to inter-cluster communications between different DCI Clusters. After that, we deeply analyze internal data flows between processes in a DCI Physical Node again. Finally, we categorize the classes of data plane acceleration frameworks.

8.1.2. Data pipeline analysis for CPS AM

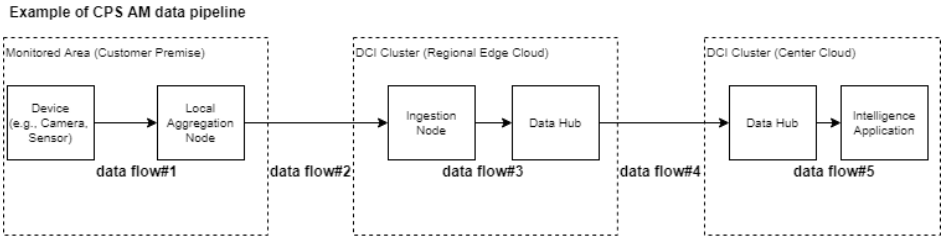


Figure 8.1-1: Example of CPS AM data pipeline

Figure 8.1-1 shows a typical data pipeline defined in Section 3 of the RIM document [IOWNGF-RIM]. In this standard pipeline, a camera device produces video streaming data continuously and uploads it to the nearest Local Aggregation node (**data flow#1**). After receiving such data from multiple cameras, the Local Aggregation node aggregates them. Then, the Local Aggregation node forwards such data to the Ingestion node (**data flow#2**). The Ingestion node receives it from multiple Local Aggregation nodes simultaneously. After receiving it from multiple Local Aggregation nodes, the Ingestion node executes processes such as conversion, decoding and resizing of the video image data, and object detection. Many applications may commonly utilize the inference results of object detection. After this process, the Ingestion node posts the output data (i.e., the inference results, and video image data) to the Data Hub for further usage (**data flow#3**). The Data Hub preserves the received data, then sends notifications to the listening nodes. In the case of maintaining the video image data, the Data Hub may store them internally. However, in the case of the inference results, they are transferred from the Data Hub in the regional edge cloud to that of the central cloud (**data flow#4**). In this case, the Intelligence Application node is supposed to subscribe to inference results, so, notifications are sent from the Data Hub to the Intelligence Application node to trigger further analysis (**data flow#5**).

data flow#1

- **Analysis**

- As defined in the RIM document [IOWNGF-RIM], the data rate of uncompressed video image is 90MB (=6 MB x 15fps) per second per camera. In the case of Motion JPEG compression scheme, the video image data rate is from 45 to 60Mbps.

- **Requirements**
  - Compatibility
    - The data flow#1 is realized on the Best Available network written in Section **Error! Reference source not found.** of the RIM document [IOWNGF-RIM]. Therefore, some processes which directly receive video image data from a camera in the Local Aggregation node should provide the first termination point of conventional network stack such as TCP/IP.

data flow#2

- **Analysis**
  - As defined in the RIM document [IOWNGF-RIM], the captured video image data in the Local Aggregation node of each Monitored Area has to be transferred to processes running in the Ingestion node located in the regional edge cloud. The bandwidth of such data transferred between the Local Aggregation node and the Ingestion node will constantly be large. For instance, when thousands of camera devices simultaneously send video images to the Local Aggregation node and the data rate of uncompressed video image is 90MB(=6 MB x 15fps) per second per camera, some processes of the Local Aggregation node have to transfer about 720Gbps of data to the Ingestion node constantly. Even if the Motion JPEG compression scheme is applied to the video data, the bandwidth will achieve 60Gbps.
  - The communication range between a Monitored Area and a regional edge cloud is assumed to be up to 337km [IOWNGF-RIM].

- **Requirements**
  - Network bandwidth
    - Likely, this data flow may require network bandwidths beyond 100Gbps.
    - Hardware-accelerated network processing is also needed, since current host CPU-based network processing is likely to become a limiting factor for achievable bandwidth. For example, research indicates 42Gbps as the maximum possible bandwidth with CPUs, along with significant energy consumption [Cai2021].
  - Message size
    - Message size of data should be configured carefully keeping three effects in mind:
      - One is that if packets per second become larger, it is well-known that performance degrades.
      - The 2nd effect is that if a mechanism to check for data transfer completion is applied over the network layer, it may lead to performance degradation especially in a long-range network. In this case, message size has significant impact to performance.
      - The last effect is that in general, using large-size messages leads to longer latency.
  - Long-range communication
    - This data flow is implemented over a long-range data plane beyond 100km.

data flow#3

- **Analysis**

- There are two types of data flow#3. One is for structured data representing the inference results (data flow#3-1). The other is related to chunked data of video images (data flow#3-2).
- Assuming that N is the number of Monitored Areas to be connected to the regional edge cloud,
  - The data flow#3-1 is estimated to be 3.6Gbits x N/second [IOWNGF-RIM].
  - The data flow#3-2 is estimated to be 60Gbits x N/second [IOWNGF-RIM].
- According to the IDH service types defined in Section 2 of the IDH document [IOWNGF-IDH],
  - “Relational Distributed Database” service type can be applied to the data flow#3-1.
  - “Object Storage” service type can be applied to data flow#3-2.

- **Requirements**
  - The requirements of these data flow#3-1 and #3-2 are almost the same as the data flow #2 except for the length of communication range.

data flow#4

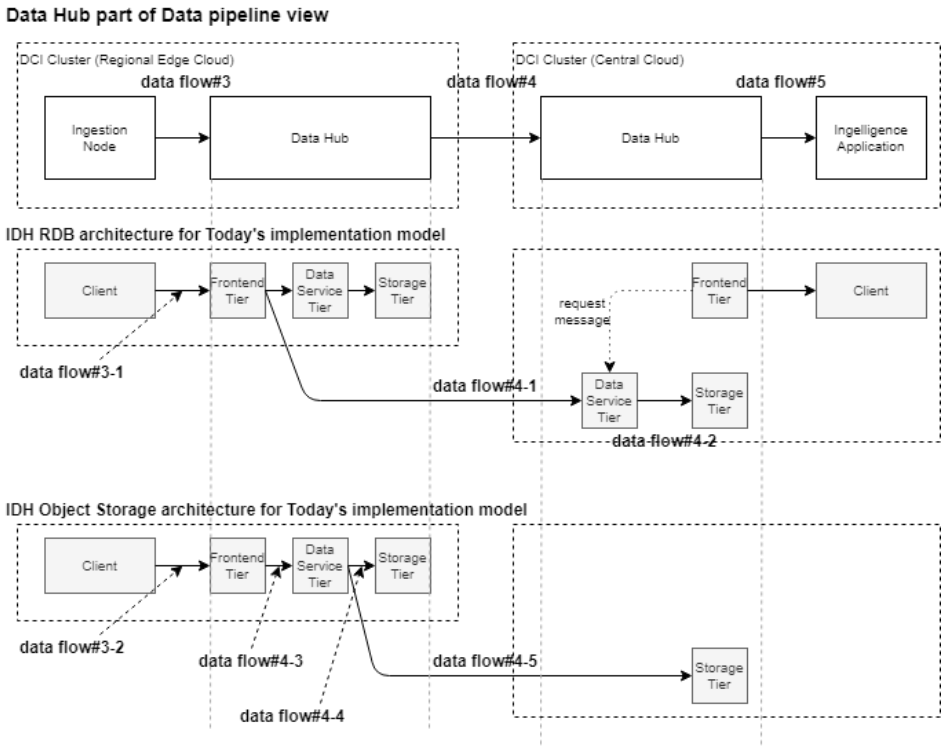


Figure 8.1-2: Example of IDH deployment for CPS AM use case

- **Analysis**
  - The data flow#4 has two types of data flow. One is relevant to the data flow#3-1 for structured data representing the result of the inference results (data flow#4-1, #4-2). The other is related to the data flow#3-2 for storing video image data(data flow#4-3, #4-4, #4-5).
  - For the structured data representing the inference results, when the Ingestion node posts it to the Data Hub placed in the regional edge cloud, it is stored in the Data Hub located in the Central Cloud (CC) depicted in Figure 8.1-2. Since the Intelligence Application node retrieves it, the Data Hub nearest to the Intelligence Application node is better to be utilized for fast access and efficient network bandwidth utilization.

        - In this case, the data flow#4-1 is implemented over the long-range data plane between the regional edge cloud and the central cloud. On the other hand, the data flow#2 is between the monitored area and the edge cloud. They are different segments and the data flow#4-1 would be longer than the data flow#2. The data flow#4-2 is propagated between Data Service Tier and Storage Tier in the same DCI Cluster. The data flow#4-1 is the mid connection, and data flow#4-2 is the backend connection defined in Section **Error! R eference source not found.** of the IDH document [IOWNGF-IDH].
    - For storing the video image data, the Ingestion node posts it to the Data Hub of the regional edge cloud. The Data Hub distributes it to the different Data Hubs via three tiers: Frontend Tier, Data Service Tier, and Storage Tier.
        - As described in Figure 8.1-2, data flow#4-3 is between Frontend Tier and Data Service Tier in the same DCI Cluster in the regional edge cloud. But there are two types of the data flow between Data Service Tier and Storage Tier (data flow#4-4, #4-5). A choice of the data flow is dependent on the use case scenario.
        - In data flow#4-3 #4-4, and #4-5, the same amount of volume as the data flow#3-2 may be required at the worst. However, when we apply load balancing algorithms for distributing data into several different IDHs, the data flow bandwidth requirement is dependent on such a scheme.

- **Requirements**
    - We assume that the number of the regional edge clouds which converge into the CC (Central Cloud) is M.
    - Network bandwidth
        - For data flow#4-1, the network bandwidth requirement is estimated to be 3.6Gbits x N x M /second.
        - For data flow#4-2, the network bandwidth requirement is also estimated to be 3.6Gbits x N x M/second. However, as Storage Tier exists in the data flow#4-2, the other requirements such as redundancy may be considered. If it has to realize the redundancy of Storage Tier, the data flow#4-2 needs to be low-latency communication in addition to high-performance throughput for storage access.
        - For data flow#4-3 and #4-4, the network bandwidth requirement is estimated to be 60Gbits x N/second at the worst. However, since this usage is for preserving the video image data, the requirements for latency and throughput can be relaxed.
        - For data flow#4-5, the network bandwidth requirement is estimated to be 60Gbits x N x M/second at the worst. However, since this usage is for preserving the video image data, the requirements for latency and throughput can be relaxed.
    - High-performance storage access
        - For the storage access in the data flow#4-2 and #4-5, its throughput should be accelerated because generally, Storage Tier is assumed to be a possible bottleneck described in Section **Error! Reference source not found.** of the IDH document [IOWNGF-IDH].
        - Backend connections between Data Service Tier and Storage Tier are expected to take place internally to the DCI Cluster. The reasons are that generally, the Data Service Tier may have to cache data of the Storage Tier and frequently write/update data that has already been stored in the Storage Tier. Further, critical messages such as for cluster heartbeat and transaction commit between these two Tiers have to be low latency [Exadata].
        - Additionally, inside the Storage Tier, new memory classes for storage, such as persistent memory, are placed in front of classical hard disk drives or flash-based drives to improve

the performance of databases and storage services. When the Storage Tier is accelerated with memory-based storage, the bottleneck point is moved to the network layer from the storage layer. Therefore, we should consider the data plane acceleration framework on Backend connections.

- From a Message Size perspective, when we look at database traffic types, these can be classified into two types of messages. One type is low latency and short-sized messages for control and management, such as cluster heartbeats or transaction commits. The other kind of message is a larger-sized message for backups and batch messages [Exadata]. Generally, control messages range from several kB to tens of kB [Exadata]. On the other hand, messages for large data should be configured to be more significant to avoid frequent ack-message exchanges.

- The communication range for each data flow is as follows:
  - The data flow#4-1 and #4-5 show long-range communication between different DCI clusters.
  - The data flow#4-2, #4-3, and #4-4 show the length between different DCI Physical Nodes.

data flow#5

- **Analysis**
  - The Intelligence Application node analyzes the labeled data gathered in the Data Hub nearest to the CC. The labeled data include a considerable amount of data produced by M x N x 1000 cameras. They are propagated over the data flow#5 between the Data Hub and the Intelligence Application node.
- **Requirements**
  - The requirement of data flow#5 is almost the same as the data flow#4-1 or #4-2 especially in terms of network bandwidth.

Common data flow features and requirements

- **Analysis**
  - In the CPS-AM use case scenario, the maximum turn-around time from the start of sensor data capture to the reception of the analysis results will be defined, e.g., 100 milliseconds ideally [IOWNGF-RIM]. This means that all of the data flows require a data plane that provides both high-bandwidth and low-latency. Furthermore, in some circumstances, data flow from the Local Aggregation node has to be duplicated to multiple destinations in the Ingestion node [IOWNGF-RIM].
- **Requirements**
  - Low-latency and low-energy consumption
    - When data is transferred between two functions or applications, a conventional network processing approach requires that the data to be transferred is copied multiple times between network and application layers. In terms of throughput and latency, this can lead to performance degradation and longer latency. Therefore, application memory areas should directly be transferred between functions or applications with low latency and high energy efficiency in a zero-copy manner [Balla2017], [Géhberger2018], [Lenkiewicz2018], [SIGCOMM2018].

8.1.3. Data plane types

Since how a data flow is implemented is dependent on data plane types, we define the three following data plane types written in Figure 8.1-3 :

- Intra-Node communication
  - Intra-Node communication is a data plane between components inside the same DCI Physical Node. This communication is conducted over an internal bus such as the PCI Express (PCIe) bus, which is a popular choice for server systems.
  - This data plane between processes inside a DCI Physical Node is standard practice.
  - It is assumed that the communication range is from a few to a few tens of centimeters.
- Inter-Node communication (Intra-Cluster communication)
  - Inter-Node communication is a data plane between DCI Physical Nodes located in the same data central. Data is propagated via Network I/O such as network interface cards, which are installed inside DCI Physical Nodes.
  - It is assumed that the communication range is from one to a few tens of meters.
- Inter-Cluster communication
  - Inter-Cluster communication is a data plane between different DCI Clusters, which are geographically located in other places. Data is also propagated via Network I/O such as network interface cards, which are installed inside DCI Physical Nodes in the same manner as Inter-Node communication.
  - It is assumed that the communication range is from a few to hundreds of kilometers.

Among these three types of the data planes, there is a natural implementation candidate technology only for Intra-Node Communication, i.e., the PCI Express, the de-facto standard of the interconnect. In contrast, there are no such candidates for Inter-Node and Inter-Cluster communication.

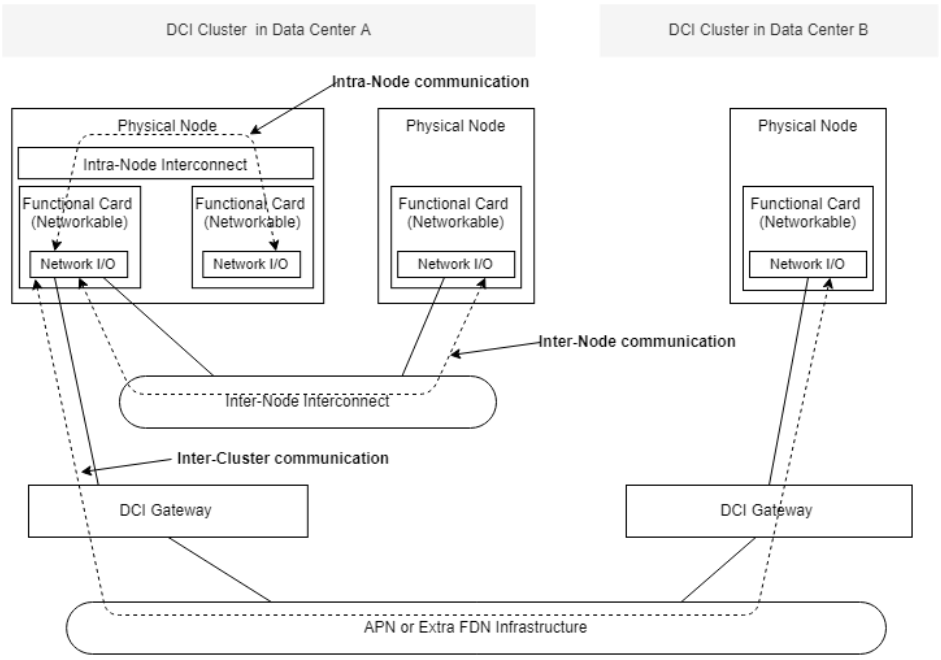


Figure 8.1-3: Intra-Node Interconnect/Inter-Node Interconnect/Inter-Cluster Interconnect

Intra-Node communication

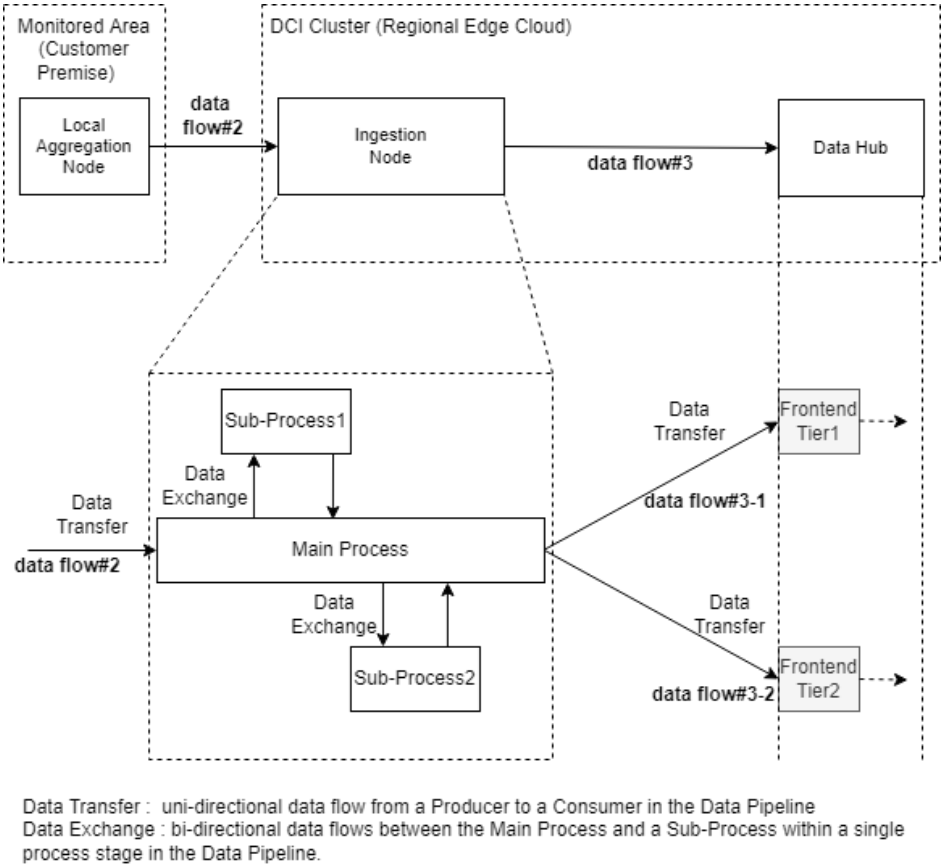


Figure 8.1-4: typical data pipeline of CPS

When we deeply analyze the data pipeline in Figure 8.1-1, we can illustrate data processing function and data flow depicted in Figure 8.1-4.

- **Main Process:** receives video image data from the camera, and then executes an inference process to detect objects via video image data after Sub-Process 1 and 2.
    - **Sub-Process 1:** Before executing the inference process, video images are decoded. The results are returned to the parent Main Process.
    - **Sub-Process 2:** Before the inference process, image resizing is executed after Sub-Process b1. The results are returned to the parent Main Process.
- **Frontend Tier1:** receives the inference results from the Main Process. This is related to the data flow#3-1 depicted in Figure 8.1-2.
- **Frontend Tier2:** receives the video image data from the Main Process for storing it. This is related to the data flow#3-2 depicted in Figure 8.1-2.

From these analyses described above, data flows can be classified into *Data Transfers and Data Exchanges*: In the case of a Data Exchange, the Main Process tends to communicate with Sub-Processes frequently with small amounts of data. Therefore, Data Exchange should be low-latency. Generally, it is implemented over Intra-Node communication. On the other hand, in the case of a Data Transfer, a single process often sends and/or receives significant amounts of bulk data.

8.1.4. Classification of data plane acceleration framework

Table 8.1-1: Class of data plane acceleration framework shows the classes for the data plane acceleration frameworks according to data plane types and data flow types. In the two classes of Intra-Node Data Transfer and Intra-Node Data Exchange, existing data plane frameworks such as DMA (Direct Memory Access) over PCIe or shared memory over PCIe are appropriate because PCIe is the most popular standard. The rest of the classes are discussed in the next section.

Section 8.2 describes the data plane acceleration framework focused on the data flow#2, #4-1 except for #4-2 which is related to storage access. On the other hand, Section 8.3 explains the data plane acceleration with storage access. A storage access part of this framework in Section 8.3 can be applied to the framework in Section 8.2.

Table 8.1-1: Class of data plane acceleration framework

DATA PLANE TYPE	COMMUNICATION RANGE	DATA FLOW TYPES	CPS-AM DATA FLOW	FRAMEWORK
Inter-Cluster	a few to hundreds of kilometers	-	#2, #4-1, #4-5	Section 8.2
Inter-Node	one to a few tens of meter	-	#3-1, #3-2, #4-2, #4-3, #4-4	Section 8.3
Intra-Node	a few to a few tens of centimeter	Data Transfer	internal process communication	existing framework, such as DMA over PCIe (Example of use case : common)
		Data Exchange	internal process communication	existing framework, such as shared memory over PCIe (Example of use case : common)
Best Available Network	-	-	#1	existing framework based on TCP/IP

8.2. Inter-Cluster Data Plane Acceleration Framework

8.2.1. Framework: RDMA over the Open APN

According to the above requirements, we propose to apply a remote direct memory access (RDMA) scheme even for long-range data flow over the All-Photonic Networks (APN) [IOWNGF-APN]. Because RDMA is generally implemented in hardware accelerated network interface cards (NIC), we expect that such NICs provide high-performance and low-latency communication by offloading kernel protocol processing and by reducing the number of memory copies (ideally in a zero-copy manner), resulting in lower CPU load. Furthermore we also expect that by providing RDMA-capable infrastructure as a default data transfer method all through end-to-end optical path in inter-node/inter-cluster communication, seamless integration of widely-distributed computing resources could be achieved without API translation between socket-based API and RDMA-based API.

8.2.2. Overview of end-to-end data plane architecture

RDMA endpoints connected by Inter-cluster interconnect over the Open APN, Extra network, and DCI are shown in Figure 8.2-1. The inter-cluster interconnect is between DCI-GW and ExtraNetwork gateway, built on top of the optical path set up by the Open APN in the diagram. The left side RDMA endpoints are inside the Extra Network in the chart, but also can exist inside the network under other DCI-GWs. Inside the ExtraNetwork and the DCI-GW, there would be communication channel provided by FlexBridge forwarding services, and such channels are chosen to be type D2 (bandwidth reserved) or D3 (bandwidth shared with congestion avoidance) depending on the use cases, referring to the Annex “IOWN Global Forum Flexible Bridge Services” [IOWNGF-APN].

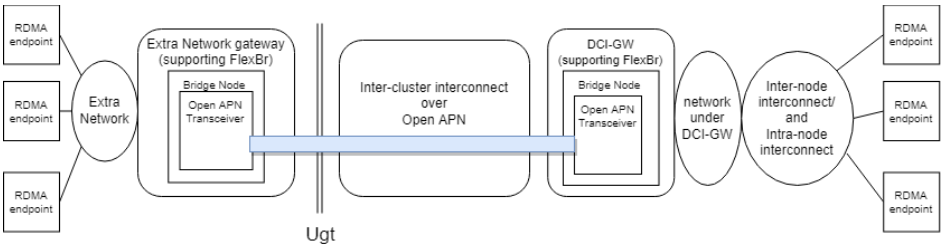


Figure 8.2-1: Inter-cluster interconnect over the Open APN

8.2.3. DPA network stack

DPA inter-cluster interconnect network stack is shown in Figure 8.2-2. The both sides inside ExtraNetwork and DCI-GW are similar, and under-layer protocols are chosen and provided by FlexBridge forwarding services. The diagram shows the RoCEv2 over Converged Enhanced Ethernet stacks to take advantage of existing stacks and existing products in the reference implementation model. Furthermore, a data flow is transferred via QoS-guaranteed layer-1 protocols with congestion-free and non-packet-multiplexed network as such Optical Transport Network (OTN) for example.

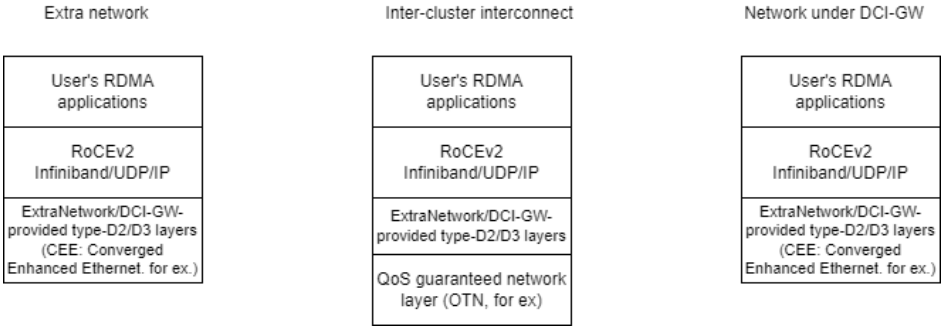


Figure 8.2-2: Inter-cluster interconnect network stack

8.2.4. Consideration

Providing RDMA functionality as an infrastructure requires consideration because RDMA is generally only applied to short-range communication, such as communication confined within a single data center. Achieving both long- and short-range RDMA transmission requires Open APN and FDN controllers to collaborate and negotiate their optimizing parameters to fulfill user’s requirements such as throughput, latency, jitter, loss rate, and so on.

Throughput degradation: Queue Depth optimization

RDMA basic operations and the issue are described in Appendix when applying it to WAN. Ack-based ping-pong control in RDMA RC (Reliable Connection) protocols may cause performance degradation in a long-range communication.

Especially a Queue Depth parameter needs to be carefully configured. The required SQ depth can be derived using the following equation, which is the same root cause as the well-known long-fat-pipe network (LFN) problems.

(RTT * LineSpeed) / MessageSize = Required QueueDepth

Because the queue depth depends on implementing different RDMA NICs, we should carefully choose proper products with enough queue depth to support the maximum distance of optical path built by the Open APN.

Efficient data transfer between RDMA NICs and FDN interface cards (HW Accelerators)

To take advantage of a zero-copy feature, RDMA NICs are better to support capabilities that enable direct data transfer between RDMA NICs and FDN interface cards in the same manner as GPU Direct [GPUD] and P2P DMA [PCI-P2P]. Or we can achieve such inter-device efficient data transfer via LSN's memories in the same manner as DMAbuf [KERNEL].

Reliability: retransmission scheme

IOWN GF use cases require very reliable underlay networks. Even though we can expect the underlying optical path provided by the Open APN to be a high-quality path with an almost zero-packet-loss feature, to guarantee the data integrity between RDMA endpoints, it should support reliable behavior such as data retransmission as defined in RDMA reliable connection (RC) scheme.

The retransmission implemented in RDMA is usually an inefficient go-back-N scheme [GBN-Wiki]**, w**hich leads to performance degradation, especially in lossy long-distance networks. As a result, an Open APN controller should set up high-quality underlying optical networks to keep the number of retransmissions to a minimum and to have a flexible RDMA implementation capable of better retransmission algorithms.

Controlling interfaces to communicate with the infrastructure orchestrator, an Open APN controller.

Collaboration with the Open APN: To optimize the queue depth described above, it is required to know the RTT between RDMA endpoints. Because the route distance of the optical path dynamically set up by the Open APN varies depending on the use cases and environment, collaboration interfaces to know the distance or RTT is required. And other than the number of RTTs, interfaces to know the characteristics of dynamically setup path includes ones for packet-loss rate, jitter, and so on. In addition, requesting interfaces for the Open APN's new optical path in a certain quality is required for the characteristics information acquisition.

QoS mapping with cooperation to the control and management plane in DCI infrastructure

DCI infrastructure functions for control, management are defined in Section **Error! Reference source not found.** of t his document. To achieve QoS aware data pipelines for IOWN GF use case, network and computing resources have to be managed in DCI's control and management plane to avoid resource contention. When the service is deployed, the relevant data flow specs are declared to DCI's control and management plane. Once the network and computing resources required for each data flow are calculated for efficient resource allocation, the control and management plane should communicate with Inter-node interconnects and DCI-GWs over the data pipeline to reserve the necessary network and computing resources. At this time, the QoS of each data flow, e.g. D2, D3, is mapped to each resource of a network interface card, physical network wired-line, Inter-node interconnects, and DCI-GWs.

8.3. Inter-Node Data Plane Acceleration Framework

8.3.1. Framework: RDMA in DCI

New storage class memory such as persistent memory is being introduced into the Storage Tier to accelerate storage access. This new storage class itself is accessed directly using memory semantics; therefore, the underlay transport network should also provide the same semantics as the new storage class to access directly. Since RDMA scheme can give this feature to network, the underlay transport network should be applied by an RDMA scheme to increase the number of achievable IOPS without any translation such as data format.

8.3.2. Overview of end-to-end data plane architecture

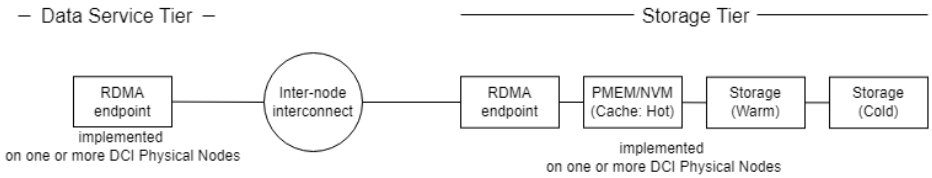


Figure 8.3-1: Example of IDH Backend connection architecture over Inter-node communication

From Inter-Node communication between Data Service Tier and Storage Tier, two RDMA endpoints read and send/write the data each other via Inter-node Interconnect defined in DCI architecture. In many cases, there are three storage tiers in the Storage Tier, Hot, Warm, and Cold storage depicted in Figure 8.3-1. The data which is accessed frequently is located in the Hot storage. Therefore, Hot storage is generally implemented in very low-latency and high-throughput storage stage. The new storage class memory is often placed in the Hot storage to act as a storage cache.

8.3.3. DPA network stack

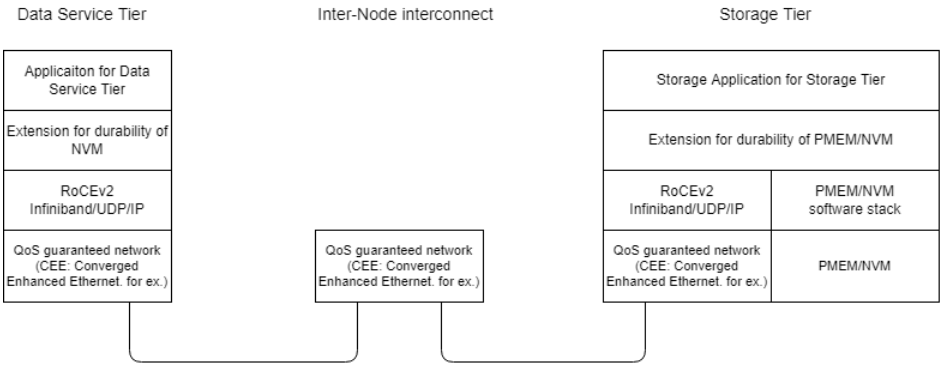


Figure 8.3-2: DPA network stack for IDH Backend connection on Inter-node communication

As mentioned before, the RDMA endpoint in the Data Service Tier is connected to that of the Storage Tier via Inter-node interconnect. Inter-Node communication between DCI Physical Nodes is implemented in DCI Cluster. RoCEv2 is a possible candidate for the underlay transport network between RDMA endpoints via the Inter-node interconnect. As multi-tenancy feature is supported in DCI Cluster, such underlay transport network is shared with multiple-types of data flows. In this sense, the technologies defined in Converged Enhanced Ethernet (CEE) specification should be applied in the underlay transport network. Inter-node interconnect should be aware of the QoS of each data flow.

8.3.4. Consideration

Data durability

Conventional RDMA cannot support data durability that is important for storage application. Even if, with the RDMA method, a memory region is directly copied into a remote memory region, this doesn't lead to immediate data durability for persistent memory. A user for the RDMA method has to execute an additional action for guaranteeing remote data persistence. Therefore, extension for conventional RDMA stack is needed. For example, after RDMA write, confirmation of whether moved memory is persistent or not needs to be executed.

Queue Depth

This should be considered in the same manner as the framework of RDMA over the Open APN as described above.

Message Size

Based on requirements, the Message Size on the RDMA layer should be configured to be small or large: we can configure up to 1GB in the case of RDMA-RC. For failure detection message of Storage Tier, low-latency message transfer is a crucial feature. In this case, the RDMA message size should be small. However, when high-performance data throughput is needed, Message size should be configured higher to avoid protocol header overhead.

Lossless network

An internal data center network is an example of an Inter-Node interconnect network. In this case, multiple different data flows produced by multiple tenants are propagated in the same network, and they have other QoS classes. Congestion control mechanism should be applied to this kind of network, e.g., Converged Enhanced Ethernet. However, there is nothing for the most appropriate mechanism. Suitable congestion control mechanisms for this are dependent on the specification of the use case.

Network Interface cache influence

Some latest network interface cards and motherboards can support incoming packet data into directly CPU cache memory without storing main memory. These technologies can provide low-latency and high-performance data throughput. However, the new storage class memory, such as non-volatile memory, requires placing the data into memory, not cache. As soon as the RDMA endpoint of the Storage Tier receives the data from that of the Data Service Tier, it should be ensured that the data is written into the new storage-class memory.

QoS mapping with cooperation to the control and management plane in DCI infrastructure

This should be considered in the same manner as the framework of RDMA over the Open APN as described above.

9. Accelerator Pooling/Sharing Frameworks

This section introduces potential methods of accelerator pooling and sharing, and describes what potential benefits these methods could bring to the DCI clusters.

9.1. Overview of technologies for accelerator pooling and sharing

More and more applications require accelerator resources to process collected data, but it is often the case that accelerators should be directly attached to physical computing nodes that host applications. Such constraints often result in operational complexity and low IT resource utilization. The accelerator pooling/sharing frameworks intend to reduce this burden by providing the ability to share unused accelerator capacity among multiple physical nodes. This would allow for the provisioning of accelerator resources that match the application's needs without having to collocate the application with the required accelerators.

Accelerator sharing is simply the ability to share the capacity of one or more accelerators with multiple workloads on the same node or on other nodes by leveraging the network or bus extension technologies. Accelerator pooling creates one or more collections of accelerators from which sharing can be implemented. Pooling is meant to provide nodes, typically in the same rack, equal access to these resources. There are two primary ways to organize accelerator sharing; dedicated pools and borrowing idle accelerators from other nodes. In either case, the ability of sharing accelerators should lead to the benefits of higher utilization and lower overall infrastructure costs.

There are two technologies to consider in order to achieve this, one is accelerator-sharing technology, which is close to the infrastructure, and the other is the application architecture which runs on top of that infrastructure. Figure 9.1-1 illustrates the methodologies of how we could place applications and accelerators. Case A depicts an elastic accelerator resource assignment to LSN. This creates a common pool of accelerator resources for a DCI cluster and assigns accelerator resources to LSNs created on the cluster. As mentioned above, there are two primary ways to achieve this, and they are shown as A-1 and A-2 in this figure. The differences and similarities are further discussed in the following sections, but CXL is one such technology and as CXL can create a resource pool only for LSNs on the same CXL bus, the framework presents methods of creating a resource pool that can be shared by LSNs on different CXL buses. Case B depicts a microservices architecture for accelerator sharing. In this case, applications are built in a microservices architecture and have microservices with accelerators process data from microservices of multiple application instances. This method brings the flexibility of the workload placement depending on its requirement, i.e., it could be used to offload general workloads from those that do not require accelerators from accelerator-attached LSNs, while those workloads which require accelerator resources could be placed close to the accelerator pool. However, the inter-microservice communication overhead may increase the workload on the contrary. Therefore, the framework presents DCI LSN compositions to achieve accelerator sharing with little inter-microservice communication overhead.

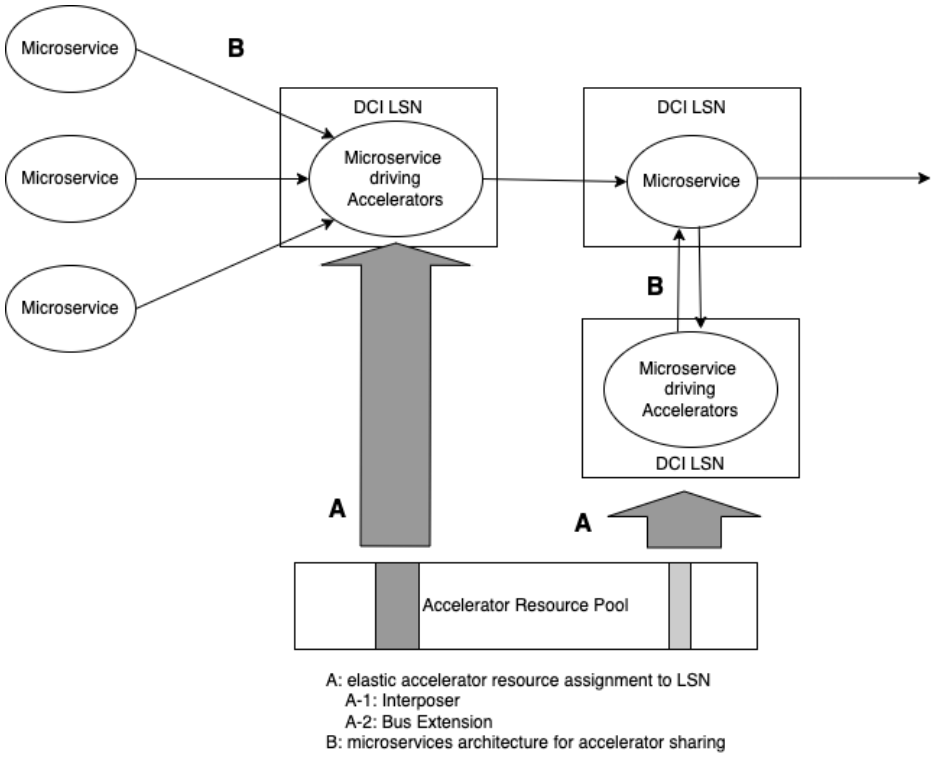


Figure 9.1-1: Example of microservices placement and accelerator pooling

In the next two sections, we will discuss the characteristics of two methodologies to achieve accelerator sharing, namely interpositioning and bus extension.

9.2. Interpositioning

9.2.1. System overview

Interpositioning utilizes the interception and remoting of specific accelerator application framework APIs.

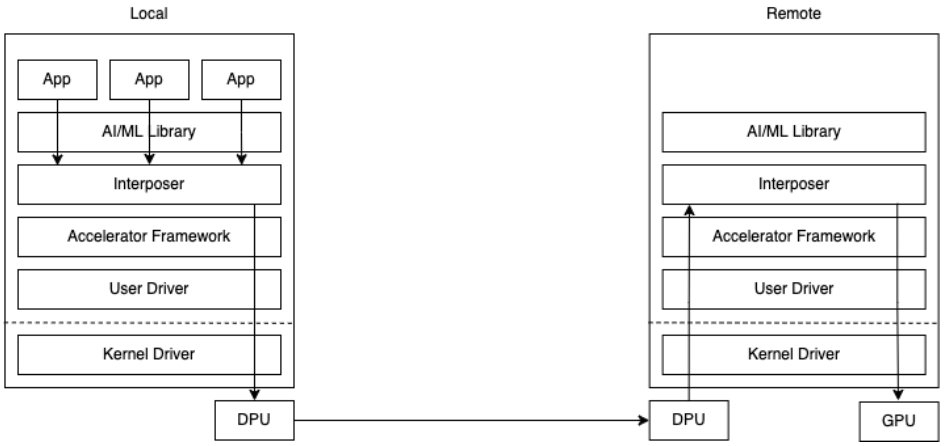


Figure 9.2-1: System diagram of Accelerator Pooling with Interpositioning

There are different layers of software between a tenant application that has the intent to utilize an accelerator and the accelerator hardware itself. Within that, there are different API boundaries that offer opportunities for the interposition of added software to virtualize the accelerator resources.

Interpositioning can occur below the kernel driver on the hardware level so the virtual resource retains the hardware interface to the accelerators. One reason that interpositioning on the driver level might be preferred over interpositioning at a higher layer is the accelerators have made it easier to provision the amount of that resource that best matches the application's needs. For example, NVIDIA's Multi-Instance GPU capabilities allow to spatially partition the resource into several fully isolated instances, which can be seen as a form of interpositioning at the hardware level.

Alternatively, as illustrated in Figure 9.2-1, the interposition can occur at higher layers, closer to the application, as in an accelerator framework or an AI/ML library. At this level, it's possible that greater liberties could be taken because the intent of the application is better known, and the interface is more software defined rather than hardware defined. One example might be to abstract the hardware choice and allow that to be chosen dynamically based on the application's performance and QoS expectations.

Another reason to interpose at a higher layer is because interpositioning at higher layers might open up more opportunities to access remote accelerators over the network that might be offered as a pool to many servers: for example, exploiting the structure of an application, work may be transferred to and from accelerators in larger batches, which may reduce transfer overhead and transmission latency impact. This level of disaggregation can help to maximize the utilization of these resources in a datacenter when, for example, a host that contains a directly attached accelerator does not have a workload that is utilizing it, but a neighboring host that does have such a workload but has no available accelerator capacity locally, could utilize the spare capacity of his neighbor's accelerator.

Utilizing remote accelerators will typically require a high-performance RDMA-capable network, such as an Open APN, so the performance costs of using that remote resource are less of an issue. Using a standardized and high performance RDMA-capable IP network is the most flexible path to be used and today's SmartNIC or DPU/IPU capabilities make this more practical.

The client-side interposition leveraged on the local side must have a server-side equivalent that operates at the same level. This creates a virtual connection between the two sides so the operations initiated on the local side can be intercepted and replayed on the remote side at that level.

Some key characteristics of application-level interpositioning frameworks that consider remote accelerators compared to frameworks that restrict the use of accelerators to those accelerators that are installed in the same server that the application software is executing on are described below:

- Pros
    - User space API client and server is highly cloud native container friendly. (API Server access to accelerator HW can use various bare-metal to full virtualization techniques to access the physical HW.)
    - Access to remote accelerators (in whole or partitions).
    - More application framework awareness allows for increased flexibility and QoS optimizations (e.g. RDMA, remote data caching, dynamic resource assignment/utilization (e.g. data path, accelerator class, local/remote, network QoS & distance).
    - Degree of performance degradation can be managed
- Cons
    - Must develop & maintain interception capabilities for potentially many frameworks & API revisions (e.g. CUDA, OpenCL, OpenMP, OpenACC) and supported programming languages.
    - Performance degradation potential

- More complex error handling - frameworks not typically built to handle remote accel access and multidimensional error scenarios & sources

9.2.2. Realization using DCI

Using DCI, example deployment is illustrated in Figure 9.2-2.

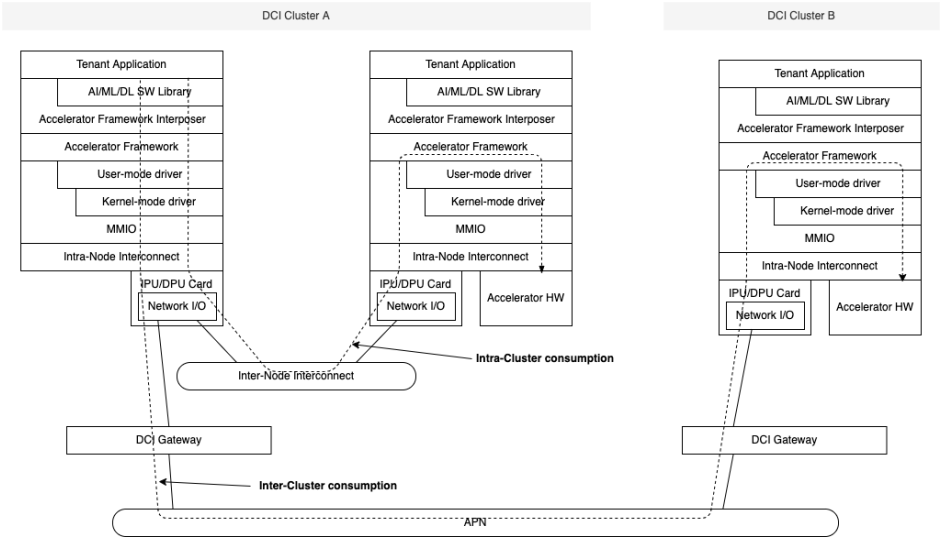


Figure 9.2-2: Example implementation diagram using interpositioning

9.3. I/O bus extension across servers

9.3.1. System overview

An additional method for pooling or sharing remote accelerator resources is referred to as I/O bus extension, which is illustrated in Figure 9.3-1. This method extends the local I/O bus (e.g., PCIe) to include devices connected to the same type of I/O bus on remote systems. This allows the operating system on one system to discover remote devices and utilize them as if they were directly connected on the local I/O bus. The techniques used to make this work can vary. ExpEther [EXPETHER], for example, utilizes a standard Ethernet for the inter-system communications data path. It relies on a special device integrated on the NIC or DPU at each end of the network. This device encapsulates the local I/O bus operations into Ethernet frames at the local (initiator) system and decapsulates them back into I/O operations on the remote (target) system.

The most anticipated I/O bus extension technique, however, will be Compute Express Link (CXL). Like ExpEther, CXL will also allow for the creation of remote pools of accelerator resources that can be discovered and utilized as if they were connected to the local system's I/O bus. In fact, CXL version 3.0 will support multi-host access to pools of accelerators, and other devices that are traditionally only accessible to the host system in which they are installed.

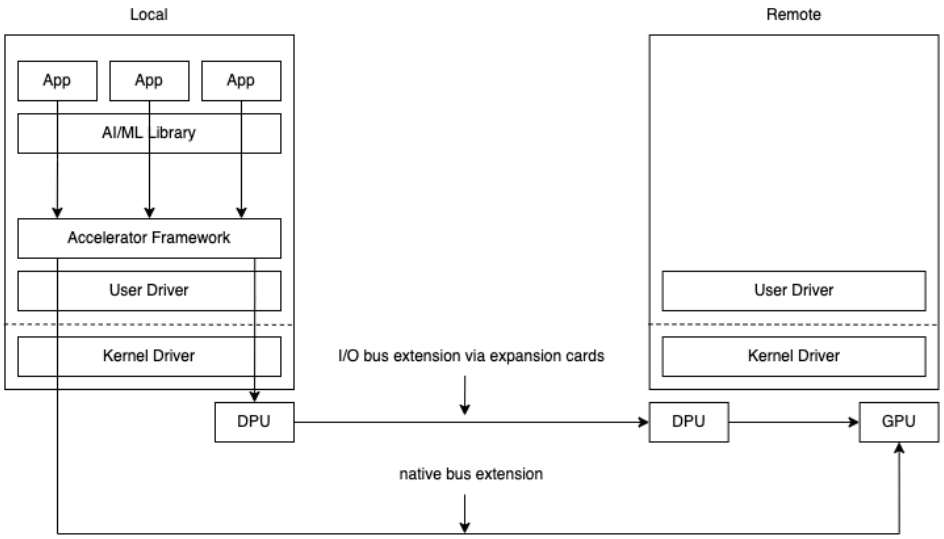


Figure 9.3-1: System diagram of Accelerator Pooling with I/O Bus Extension

9.3.2. Realization using DCI

Using DCI, example deployment is illustrated in Figure 9.3-2.

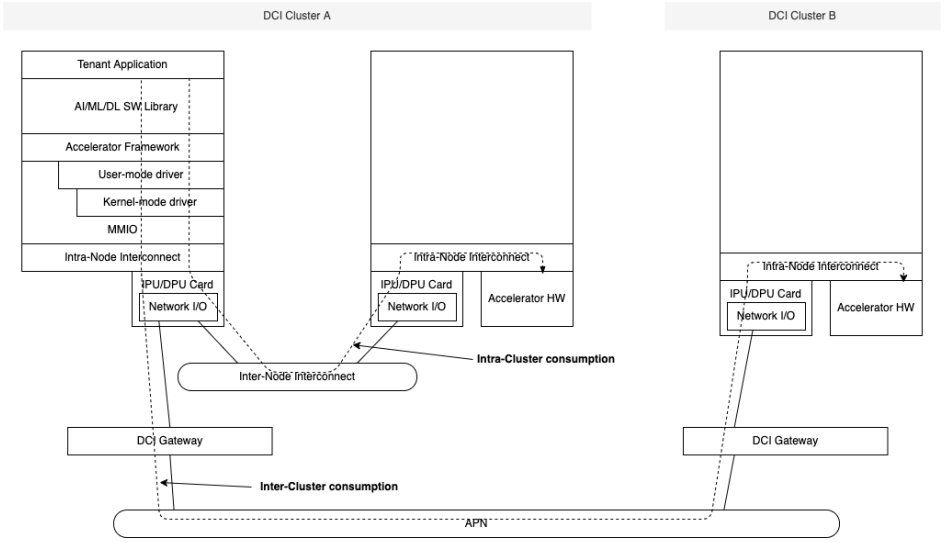


Figure 9.3-2: Example implementation diagram using I/O Bus Extension

9.4. Microservices

9.4.1. Overview

We can achieve accelerator sharing by building an application in a microservices architecture. As illustrated in Figure 9.4-1, in this method, we deploy multiple microservices, called client microservices, that send data processing requests to one microservice, i.e., serving microservice.

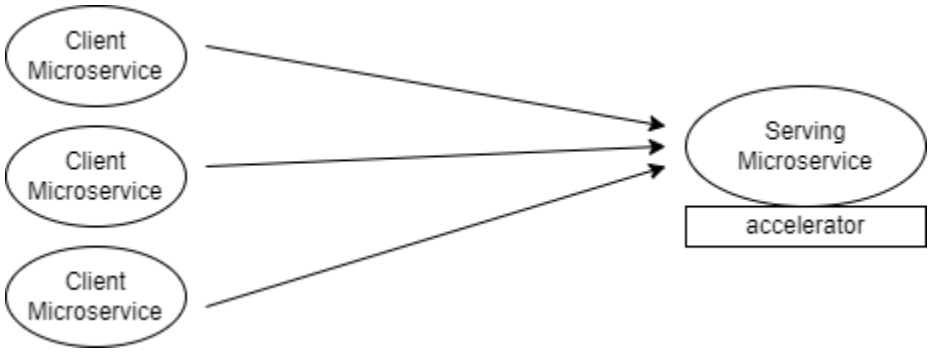


Figure 9.4-1: Example implementation diagram using I/O Bus Extension

There are several design choices to achieve this goal.

First, the serving microservice may be application-specific or common to multiple applications. Deploying an application-specific serving microservice would help us achieve high efficiency because we can implement a serving microservice that executes a sequence of multiple tasks upon request. On the other hand, in the latter case, we can adopt the serving framework of major ML frameworks such as TensorFlow and pyTorch. This would create opportunities for infrastructure service providers to provide Accelerator-as-a-Service.

Second, data processing requests may be in one-way messages or in round-trip, i.e., request-and-reply, messages. In the case of one-way messages, the serving microservice should forward the results of data processing to another microservice. This would help us implement a client microservice that handles a large volume of data at a constant rate. On the other hand, round-trip-based implementations may be regarded as a natural choice for application developers that are familiar with gRPC. However, one of the issues with round-trip-based implementations is that their performance may degrade significantly with the network latency.

Third, there are many design choices as to the request protocol and data transfer method. For example, we may use RDMA to achieve efficient data transfer.

9.4.2. Major use-cases for pools based on microservices

Hybrid Cloud

While many enterprises want to build and operate AI systems to advance their businesses, they are under strong pressure to reduce carbon emissions. They will be able to cope with this situation by using service providers' up-to-date accelerators on an as-a-service (aaS) basis.

Edge Computing

Edge computing use-cases are those use-cases that require high bandwidth and low latency to fixed geographical locations. Here, the speed of light itself becomes the ultimate barrier that prohibits deploying such applications in distant regional datacenters or central hyperscale datacenters. Instead, edge computing applications are typically executed in specialized small-scale edge datacenters that fulfill the high QoS requirements of their respective use-cases.

However, edge datacenters typically operate at lower cost-efficiency per offered performance, since edge datacenters cannot exploit the economies of scale as well as regional or central datacenters can. This is particularly true for applications that must be executed on servers with expensive, but also highly efficient specialized accelerator hardware.

One way to combine both the computational efficiency of accelerator hardware and the economical efficiency of regional and central datacenters is to decompose monolithic applications into microservices, abstracting the communication between datacenters. The freedom gained by doing so can then be used to independently place each microservices in the most efficient datacenter that still fulfills the QoS requirements of the overall application.

This way, applications decomposed in this manner can then transparently make use of accelerators located inside pools, such as pools that are local to a given edge datacenter, within a near-edge datacenter, or farther away in regional or even central datacenters as far as the latency constraints of an application allow. As a result, the utilization of all installed accelerators can be improved.

Accordingly, the lower the communication overhead of the microservices is, the more freedom an orchestration system has to allocate resources and deploy applications, and therefore, the higher the economical benefit is. The following section outlines how such inter-microservices overhead can be reduced dramatically by the use of IOWN GF Open APN and DCI technologies.

9.4.3. Realization using DCI

This subsection presents DCI LSN compositions to achieve the presented accelerator-sharing method with little inter-microservice communication overhead.

Application-Specific Serving Microservice & One-Way Messages: Transferring data through RDMA over APN will help us reduce the inter-microservice communication overhead. Hence, each microservice should run on a DCI LSN with an IPU/DPU card or RDMA NIC as shown in Figure 9.4-2. One of the remaining design issues is how to create pairs of memory regions (MR) for RDMA transfer. This depends on the applications. Some applications may dynamically create MR pairs using a control channel between the microservices.

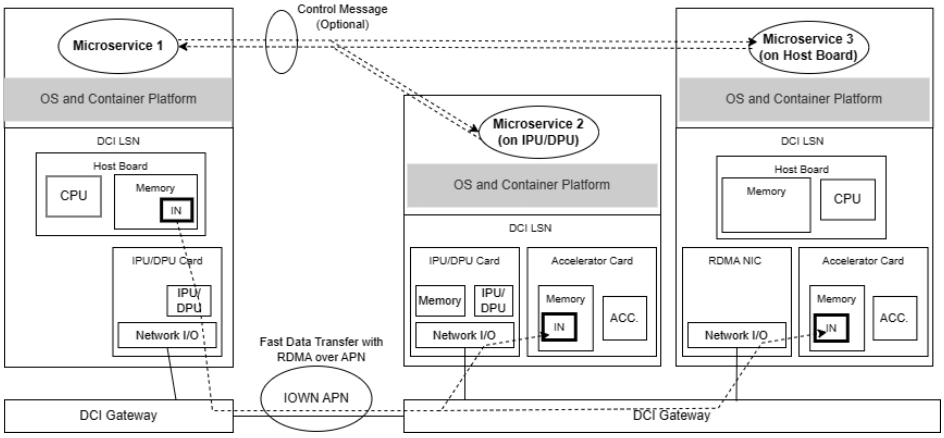


Figure 9.4-2: One way messaging in case of application-specific serving microservice

Common Serving Microservice & Round Trip Messages: Adding IPU/DPU cards to DCI LSNs which is shown in Figure 9.4-3 would help us reduce the inter-microservice communication overhead because, for example, these cards offload the network virtualization overhead from the host CPU. IPU/DPU cards may be able to do more to improve the efficiency of gRPC communication. This is a subject for further study.

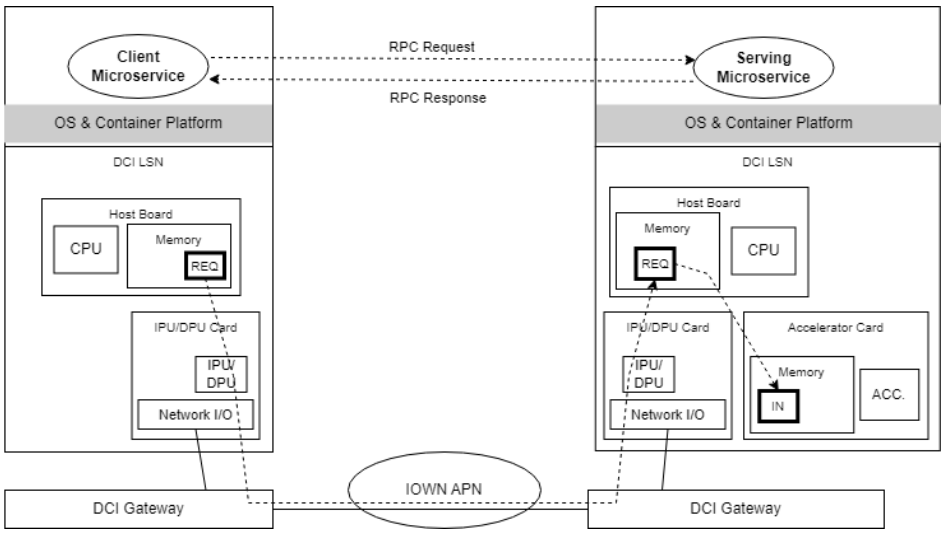


Figure 9.4-3: Round trip messaging in case of common serving microservice

9.5. Comparison of the discussed approaches

We have discussed how accelerators could be shared across physical hosts as well as across applications. These two could be rephrased as two layers as they are independent of each other and each has its own design patterns:

Accelerator sharing design patterns:

- Direct attach, local accelerator
- Interpositioning
- I/O Bus Extension

Application design patterns:

- Standalone
- Pipelining
- Serving

Designers and developers could choose any combinations from these two layers depending on their application and infrastructure requirement. Below are two example combinations that are highlighted to show how they could be combined for specific use cases.

1. Serving application with accelerator sharing through I/O Bus Extension (CXL in this case)

    To accelerate the adoption of IOWN-GF to a wider audience, it's vital to use widely accepted frameworks. A few major AI frameworks (e.g. PyTorch, TensorFlow) have large developer communities. These frameworks regularly unlock the full potential of PCIe-based accelerators today. Once CXL (an emerging industry standard) is as ubiquitously deployed as PCIe, and accelerators support it, the AI frameworks will no longer be limited to using higher layer software services, such as interpositioning, in order to match acceleration demand with accelerator capacity. An added bonus is the frameworks and the many existing use cases they support should work as is, provided the CXL device composition and management is addressed elsewhere.

2. Standalone application with direct attach accelerator

    The idea of accelerator sharing and pooling is based on the optimization of the low accelerator utilization rate. Hence, it's natural to use the locally attached accelerator if the application has an extremely low latency

requirement for its results. An application performing AI inferencing on a constant stream of data would be a good example of this combination.

Two application design examples illustrating function placement with pipeline and serving approaches are shown in Figure 9.5-1.

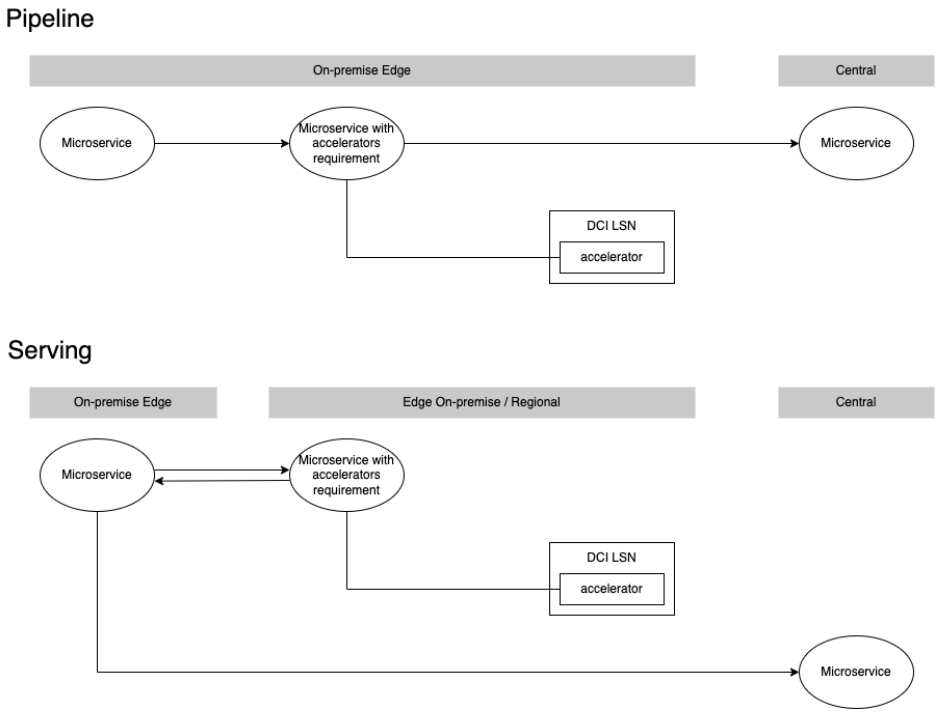


Figure 9.5-1: Example of function placement

10. Conclusion

In this technical report, we described the DCI system architecture and its functional components, along with the DCI service model, the system procedures, and the concept for data plane acceleration. The use cases of IOWN GF for mobile networks and cyber-physical systems are used as examples to illustrate the DCI deployment model. To understand why the data plane acceleration framework is essential for the IOWN GF use cases, the data pipeline derived by the use case's scenario of area management in the cyber-physical systems is analyzed.

DCI, Open APN, and FDN together form the communication and computing infrastructure of the IOWN GF System. The primary function of the DCI subsystem described in this report is to provide the IOWN GF infrastructure as a service: LSNs can be launched on DCI clusters on demand. This enables highly flexible, scalable, and efficient computing for the various use cases and deployment scenarios that the IOWN GF envisions. The framework of "RDMA over the Open APN" accelerates data transfer for the long-range data flow over the APN. On the other hand, the framework of "RDMA in DCI" provides high-performance data transfer in the DCI Cluster.

Moving forward, as IOWN GF continues to refine its services and requirements, more details on the DCI are to be defined. IOWN GF DCI work will co-develop with other IOWN GF working areas to refine the DCI architecture and enable additional features and services.

Abbreviations and acronyms

For the purposes of this Reference Document, the following abbreviations and acronyms apply:

ABBREVIATION	DESCRIPTION
AI	Artificial Intelligence
AIC	AI-Integrated Communication
AM	Area Management
API	Application Programming Interface
APN	All-Photonic Network
C/A	Connector/Adaptor
CC	Central Cloud
CD	Connection Description
CEE	Converged Enhanced Ethernet
CNF	Cloud-native Network Function
CPS	Cyber-Physical System
CPS-AM	CPS Area Management
CPU	Central Processing Unit
CRUD	Create, Read, Update, and Delete
CU	Central Unit
CUDA	Compute Unified Device Architecture
CXL	Compute Express Link
DCI	Data-Centric Infrastructure
DCICC	DCI Cluster Controller
DCIIO	DCI Infrastructure Orchestrator
DCIOC	DCI Object Composite
DCI-GW	Data-Centric Infrastructure - Gateway
DII	DCI Inter-node Interconnect
DPA	Data Plane Acceleration
DPU	Data Processing Unit
DU	Distributed Unit
EVC	Ethernet Virtual Connection
FDC	Function-Dedicated Computing

FDN	Function-Dedicated Network
FPGA	Field Programmable Gate Array
GF	Global Forum
GPU	Graphics Processing Unit
gRPC	gRPC Remote Procedure Call
IaaS	Infrastructure-as-a-Service
IDH	IOWN Data Hub
IN	Ingestion
IOWN	Innovative Optical and Wireless Network
IOWN GF	IOWN Global Forum
IP	Internet Protocol
IPU	Infrastructure Processing Unit
I/O	Input/Output
IT	Information Technology
LA	Local Aggregation
LAN	Local Area Network
LSN	Logical Service Node
MIG	Multi-Instance Graphics Processing Unit
ML	Machine Learning
MNO	Mobile Network Operator
MPLS	Multi-Protocol Label Switching
NIC	Network Interface Card
OpenACC	Open Accelerators
OpenCL	Open Compute Language
OpenMP	Open Multi-Processing
PCIe	Peripheral Component Interconnect Express
QoS	Quality of Service
RAN	Radio Access Network
RDMA	Remote Direct Memory Access
RIM	Reference Implementation Model
RoCEv2	RDMA over Converged Ethernet version 2
RTT	Round Trip Time

RU	Radio Unit
SDI	Serial Digital Interface
SDK	Software Development Kit
SmartNIC	Smart Network Interface Card
SMO	Service Management and Orchestration
SQ	Send Queue
SR-IOV	Single Root Input/Output Virtualization
TSN	Time-Sensitive Networking
VNF	Virtualized Network Function
VPU	Vector Processing Unit
vRAN	Virtual Radio Access Network
WAN	Wide Area Network
WI	Work Item
ZTP	Zero-Touch Provisioning

References

REFERENCE	DESCRIPTION
[IOWNGF-APN]	IOWN Global Forum, “Open All-Photonic Network Functional Architecture,” 2022
[IOWNGF-DCIaaSPoC1]	IOWN Global Forum, “Data-centric-infrastructure-as-a-service PoC Reference”, 2022
[IOWNGF-IDH]	IOWN Global Forum, “Data Hub Functional Architecture,” 2022
[IOWNGF-IMN]	IOWN Global Forum, “Technical Outlook for Mobile Networks Using IOWN Technology,” 2022
[IOWNGF-RIM]	IOWN Global Forum, “Reference Implementation Model (RIM) for the Area Management Security Use Case,” 2022
[MEF3PoC]	”MEF 3.0 Proof of Concept (138) - E2E Slicing for Extreme Services,” The Okinawa Open Laboratory and NTT Group developed a proof concept system of the slice orchestration among RAN, Carrier Ethernet, and Optical Transport, February 2021. https://youtu.be/i5NnmsagisA
[Cai2021]	Qizhe Cai, Shubham Chaudhary, Midhul Vuppalapati, Jaehyun Hwang, and Rachit Agarwal, “Understanding host network stack overheads,” Proceedings of the 2021 ACM SIGCOMM 2021 Conference (SIGCOMM '21), August 2021, Association for Computing Machinery, New York, NY, USA, 65–77. DOI:https://doi.org/10.1145/3452296.3472888
[Exadata]	Introducing Exadata X8M: In-Memory Performance with All the Benefits of Shared Storage for both OLTP and Analytics, https://blogs.oracle.com/exadata/post/exadata-x8m
[Balla2017]	Dávid Balla, “Minimizing the latency of the RDMA based communications,” 2017, https://tdk.bme.hu/VIK/DownloadPaper/sdf9
[Géhberger2018]	D. Géhberger, D. Balla, M. Maliosz and C. Simon, "Performance Evaluation of Low Latency Communication Alternatives in a Containerized Cloud Environment," 2018 IEEE 11th International Conference on Cloud Computing (CLOUD), 2018, pp. 9-16. DOI: 10.1109/CLOUD.2018.00009.
[Lenkiewicz2018]	Przemyslaw Lenkiewicz, P. Chris Broekema, Bernard Metzler, Energy-efficient data transfers in radio astronomy with software UDP RDMA, Future Generation Computer Systems, Volume 79, Part 1, 2018, pp.215-224, ISSN 0167-739X
[SIGCOMM2018]	“Topic preview,” SIGCOMM 2018, https://conferences.sigcomm.org/sigcomm/2018/files/tp/sigcomm18-tp-06-rdma-and-hardware-support.pdf

[GPUD]	"Enhancing Data Movement and Access for GPUs," https://developer.nvidia.com/gpudirect
[PCI-P2P]	"PCIe Peer-to-Peer (P2P)," https://xilinx.github.io/XRT/master/html/p2p.html
[KERNEL]	"Buffer Sharing and Synchronization," The Linux Kernel documentation, https://www.kernel.org/doc/html/latest/driver-api/dma-buf.html
[GBN-Wiki]	"Go-Back-N ARQ," Wikipedia, https://en.wikipedia.org/wiki/Go-Back-N_ARQ
[EXPETHER]	ExpEther Consortium - Technology https://expether.org/technology.html

Appendix A: RDMA operation and performance considerations

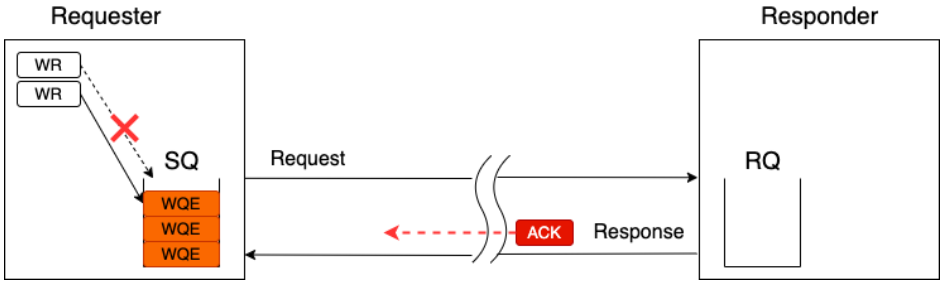


Figure A-1: Queue Pair operation for basic RDMA scheme

RDMA operation is managed by using Work Queue Entry (WQE), which includes information of memory address and data length to be sent to the RDMA memory region of the transfer destination host. RDMA provides several types of communication models, e.g., reliable connection (RC), unreliable datagram (UD), etc.

In the reliable connection mode, RDMA protocols implement retransmission and flow control mechanisms to guarantee loss-less communication. Loss-less communication is typically implemented by a ping-pong acknowledgement messages-based control protocol, similar to the TCP protocol. To clarify why performance optimization is required, the overall data transmission behavior of the RDMA Reliable Connection mode is illustrated in Figure A-1 and is described below;

- First, at the sender side (requester), a WQE is enqueued into the Send Queue (SQ) implemented within the hardware of an RDMA NIC. This enqueue operation triggers and therefore starts the data transmission from the requester to the responder.
- After that, at the receiver side (responder), right after the completion of data reception, an acknowledgement (ACK) message is sent back to the sender side to notify the completion of data reception.
- Finally, at the sender side (requester), receiving the ACK triggers to complete the already sent data by dequeuing the WQE from the SQ.

Therefore, considering the above, when using the RDMA Reliable Connection mode in a WAN environment, longer RTT will lead to long delays until ACK messages are received at the sender (requester) side. In turn, this reception delay results in long delays until WQEs that are enqueued in the SQ can be dequeued. The SQ size and the number of WQEs that can be stored within is limited: therefore, if SQ is full with remaining WQEs, it can no longer enqueue new WQEs until more ACK messages are received. To summarize, this effect impacts data transmission throughput for high data rates under long link transmission delays.

The RDMA unreliable datagram (UD) mode does not have such an ACK-based mechanism and just sends and receives data without acknowledgment in the same manner as the UDP protocol. Therefore, the RDMA UD mode does not suffer performance penalties over links with high transmission delay in the same way as described above; however, the use-case for the RDMA unreliable datagram mode is very limited.

Appendix B: Further clarification of DCI concepts

This appendix further clarifies central DCI concepts: Below, first Appendix B.1 further elaborates on the nature of logical service nodes (LSNs). After that, Appendix B.2 discusses various DCI Cluster implementation strategies. Finally, Appendix B.3 presents an example of an instance of the DCI cluster model.

B.1 Relation between LSNs, applications, and application functional nodes

This appendix further illustrates how logical service nodes (LSNs) as introduced in Section 1.3, hardware pools, the DCI service exposure function, user applications, and application functional nodes are related to each other. Below, first LSNs are further described in Appendix B.1.1, and after that, example LSN usage scenarios are presented in Appendix B.1.2.

B.1.1 Further description of LSNs

Role-wise, LSNs primary correspond to bare-metal servers and are a concept at the physical or hardware level. The key defining point of LSNs is that their components, i.e., Functional Cards, can natively communicate with each other using memory semantics. Following, LSNs are more generic than hardware servers that mandatorily include a complete mainboard, since LSNs may also consist only of individual extension cards connected by a bus fabric without requiring a separate mainboard as typical bare-metal servers do. It is expected that in typical IOWN GF systems, LSNs are created from hardware pools.

Since LSNs are a concept at the hardware level, hypervisors, virtualization software, orchestration frameworks, and similar software is executed as software provided by DCIaaS users or tenants on top of LSNs. However, LSNs may also be created from hardware components that inherently support logical partitioning of hardware or certain forms of hardware virtualization, such as SR-IOV, MIG, or CPU hardware threads, as long as the fact that hardware is logically partitioned or virtualized is not visible to the outside of the DCI Cluster and in particular not exposed to the software executing on top of LSNs.

Finally, application functional nodes are comprised by software executing on top of the LSNs, and may comprise multiple LSNs.

B.1.2 Example LSN usage scenarios

Examples of how scenarios of how LSNs could be employed and how LSNs are related to other concepts are shown in Figure B-1.

LSNs are viewed differently from two domains: Users of a DCIaaS system request creation, update, and destruction of LSNs via the “S” DCI service exposure function interface. Together with an LSN hardware specification, users also provide initialization information for parts of the LSN, such as OS boot images. Software executed on top of LSNs may include, for example, bare-metal real-time applications, virtual machine hosting software, or orchestrator worker nodes.

The actual creation, updates, and destruction of LSNs occurs on the infrastructure layer that is completely hidden from DCIaaS users behind the “S” DCI service exposure function interface. Within this infrastructure, various DCI managers, orchestrators, and controllers control and configure the hardware of DCI Clusters to create LSNs. It is expected that many DCI Cluster implementation approaches internally create LSNs from hardware pools.

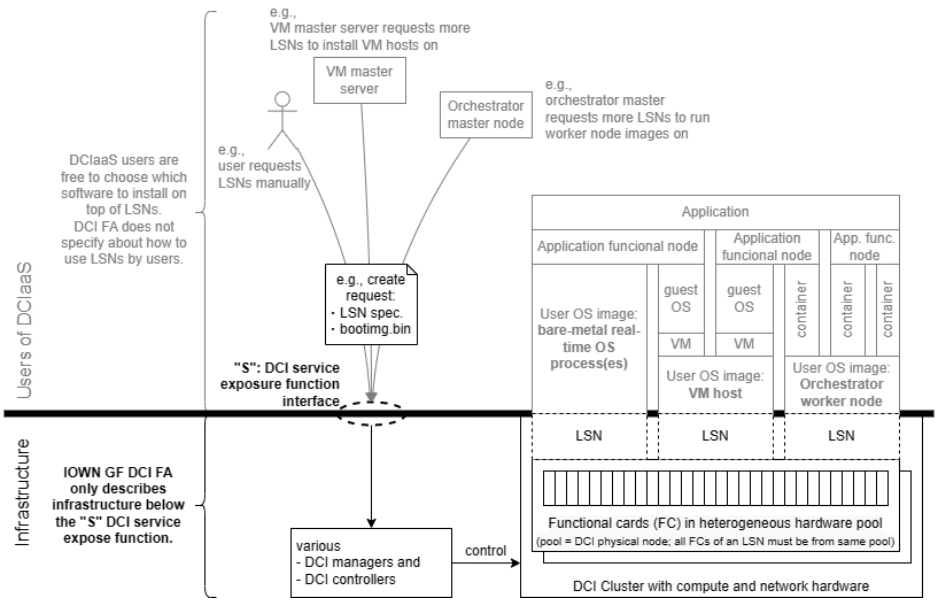


Figure B-1: Generic LSN usage patterns

B.2 DCI Cluster implementation strategies

This appendix introduces three possible DCI Cluster implementation strategies: First, the most straight-forward realization method is to prepare classic servers corresponding to LSNs in advance as outlined in Appendix B.2.1. After that, a possible system structure for implementations that use dedicated extension cards to virtually extend system buses beyond its original domain such as mainboards by tunneling over Ethernet in Appendix B.2.2. Then, the option of using specialized bus fabrics that reach beyond a single server or mainboard are visited in Appendix B.2.3. Finally, as an extreme example, the on-the-fly construction of DCI Gateways from hardware pools is explored in Appendix B.2.4.

While the first three approaches appear to be the most promising, hardware providers are free to invent and employ other approaches such as illustrated in Appendix B.2.4, as long as DCI Clusters making use of these approaches can behave according to the DCI Cluster model that is under specification (cf. Section 7.3). Furthermore, implementors may choose to employ multiple approaches in parallel in the same DCI Cluster.

B.2.1 Server power on/off control

The most straight-forward way to realize a DCI Cluster with the ability to issue LSNs to its users is to prepare the required computers to implement LSNs in advance. This approach is illustrated in Figure B-2. In this case, the creation of an LSN corresponds to initializing or powering on such a pre-configured server. In turn, the destruction of an LSN then corresponds to powering off such a server. Further, changing LSN configuration is not supported in this case.

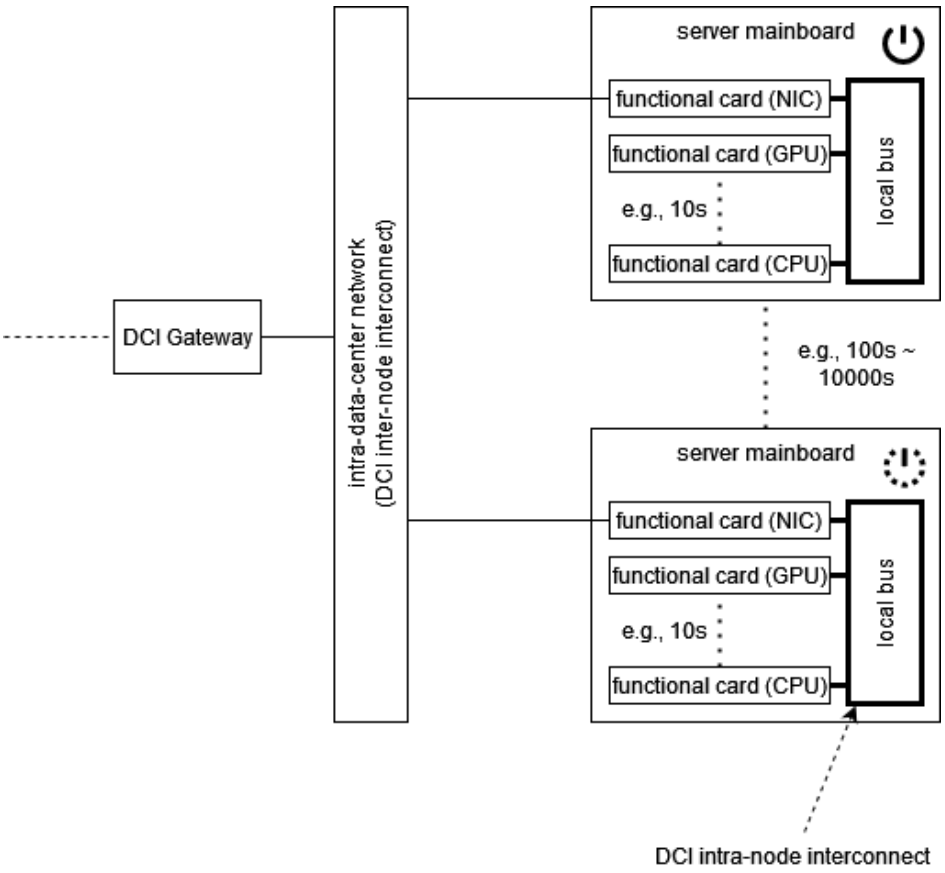


Figure B-2: Example of DCI Cluster based on server power control

In terms of the IOWN GF DCI Functional Architecture, the local buses on the server mainboards correspond to separate DCI intra-node interconnects, and individual servers as a whole correspond to separate DCI Physical Nodes.

Although a very straight-forward method, this approach might see practical use when the nature of particular LSNs is so special that further breaking up LSNs into their components and managing these LSNs in pools is not sensible due to technical or economical reasons, but such LSNs still need to be integrated into DCI Clusters. Examples could include atomic clocks, controllers to specialized or unique opaque hardware, or physically encapsulated crypto hardware.

For further information, readers are referred to the DCIaaS PoC Reference [IOWNGF-DCIaaSPoC1], Sec. 4.1.2.

B.2.2 Bus extension via Ethernet

Another implementation approach for DCI Clusters is to virtually extend and connect server mainboard system buses via tunneling over networks such as Ethernet. This approach increases the flexibility of instantiating LSNs by allowing multiple classical server mainboards with separate system buses to share hardware devices with each other via this bus extension mechanism. This approach is illustrated in Figure B-3.

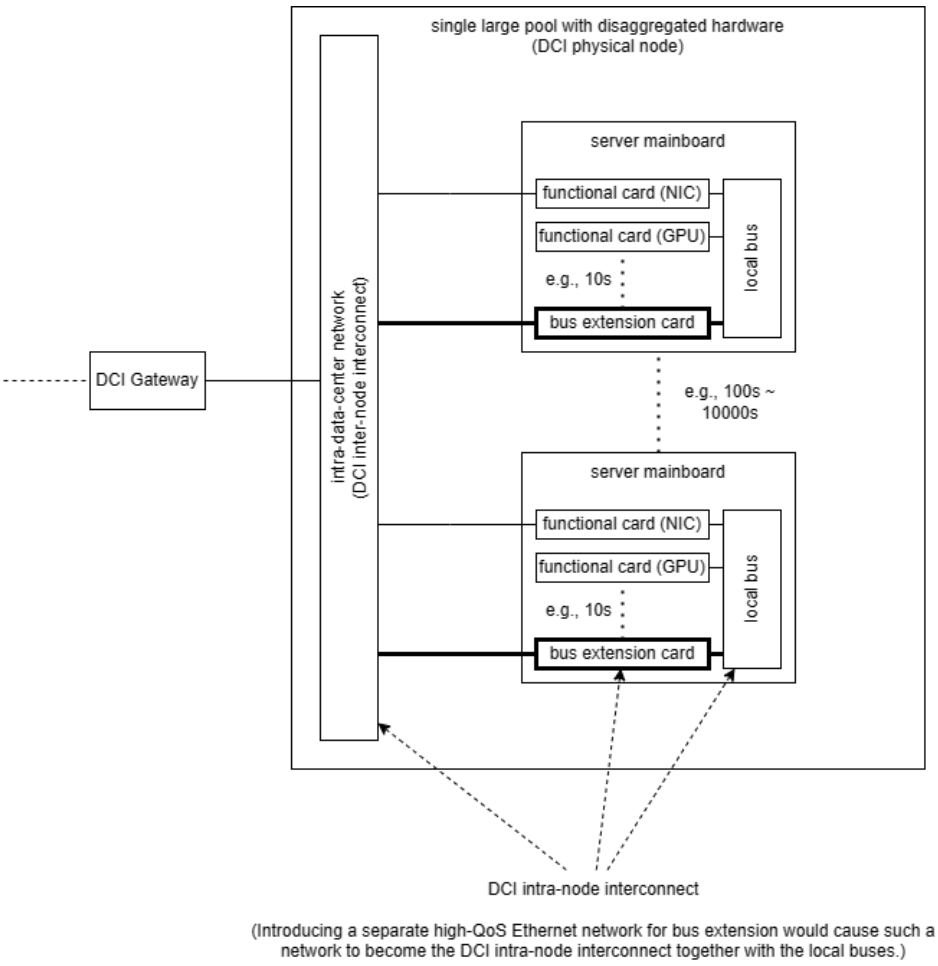


Figure B-3: Example of DCI Cluster based on bus extension via Ethernet

In terms of the IOWN GF DCI Functional Architecture, the local buses of the server mainboards, the bus extension cards, and the network the bus extension cards are connected to together correspond to the DCI Intra-node interconnect. Analogously, all server mainboards whose bus extension cards can communicate with each other taken together and able to share their resources from a single DCI Physical Node. Additionally, the network that the network interface cards are connected to corresponds to the DCI inter-node connect. Finally, all server mainboards behind the DCI Gateway are connected to the same DCI intra-node interconnect (realized by the bus extension cards connected to the same network), therefore, this single large DCI Physical Node corresponds to a DCI Cluster.

Furthermore, including the intra-datacenter-network in the intra-node interconnect directly follows from that LSNs are created components connected to the same intra-node interconnect. If both bus extension cards and also network interface cards are connected to the same physical network infrastructure as illustrated in Figure B-3, then this network realizes both the DCI inter-node interconnect as well as partly the DCI intra-node interconnect at the time. However, the discussion of feasible network topologies for the realization approach of this section is beyond the scope of this document.

B.2.3 Specialized bus extension fabrics

The implementation approach for DCI Clusters highlighted in this appendix is to employ a specialized bus extension fabric to connect server mainboard system buses with each other in a native manner. This approach is illustrated in Figure B-4. While in principle able to provide similar benefits as the approach to extend the system bus over a non-

native network via bus extension cards, differences that might emerge are relating to the number, types, and scale of the physical interconnects required.

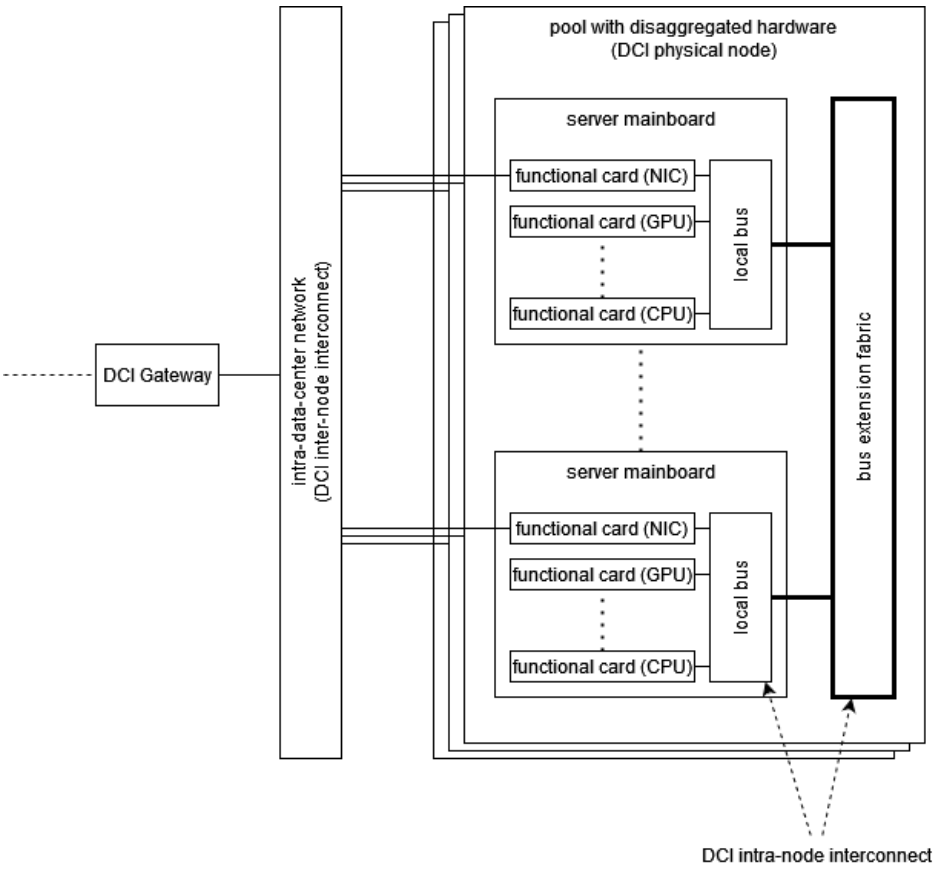


Figure B-4: Example of DCI Cluster based on specialized bus extension fabric

In terms of the IOWN GF DCI Functional Architecture, the local buses of the server mainboards together with the bus extension fabric form the DCI intra-node interconnect. Analogously, all server mainboards connected directly through their local buses to the same bus extension fabric together form a single DCI Physical Node. Finally, the intra-data-center network, e.g., the regular Ethernet network, corresponds to the DCI inter-node interconnect.

Furthermore, selecting a non-Ethernet fabric to realize the DCI intra-node interconnect implies that there exist specific requirements that Ethernet networks cannot meet or cannot meet with sufficient cost-efficiency. Such requirements could be latency, jitter, packet loss probability, or bandwidth requirements. In case of latency requirements, the sensible physical dimensions of a single DCI Physical Node may be constrained by light-speed transmission delays. This might lead, for example, to multiple separate DCI Physical Nodes existing in parallel. Another possibility would be the introduction of large hierarchical interconnects. However, analogous to the bus extension approach illustrated in Appendix B.2.2, the discussion of feasible network topologies for the realization approach of this section is beyond the scope of this document.

B.2.4 Software-defined DCI Gateway

The last approach to construct a DCI Cluster shall serve to illustrate the freedom that vendors have to construct DCI Clusters. Here, as an extreme example, the DCI Gateway does not exist as a separate hardware unit, but is constructed from the same hardware pools from which logical service nodes are constructed. This approach of *software-defined DCI Gateways* is illustrated in Figure B-5: Each hardware pool is equipped with none, one, or multiple NICs, IPUs, or DPUs that are equipped with long-range transceivers. These transceivers directly connected to APN fibers, for example,

via fiber patch cords or layer 1 optical switches. The functional cards mounting such transceivers could perform the main interactions with the APN on their own, or alternatively, act as simple interfaces of a switch- or router-like server that is composed from the hardware pool components. Such servers can therefore also include arbitrary elements beyond network-interfacing cards such as CPUs, GPU, FPGAs.

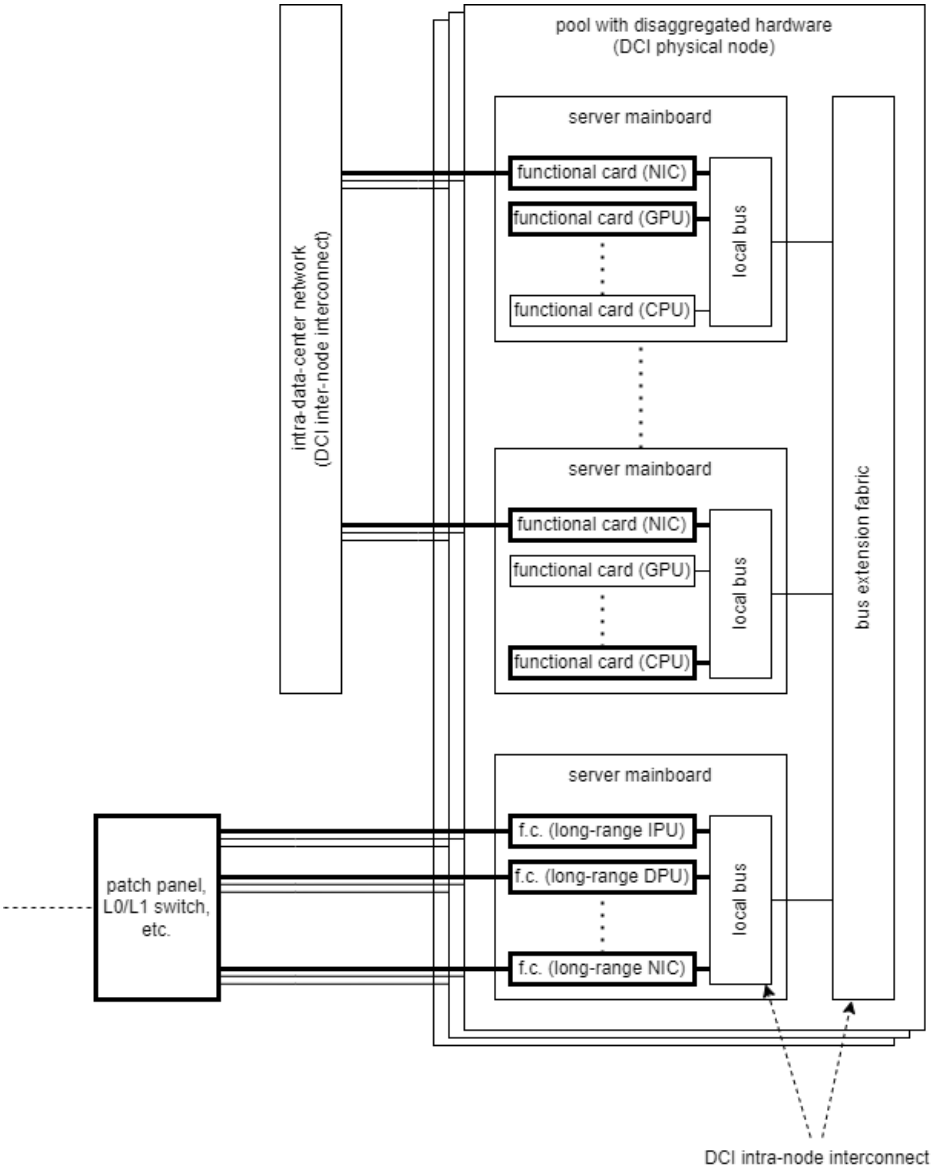


Figure B-5: Example of DCI Cluster without dedicated DCI Gateway

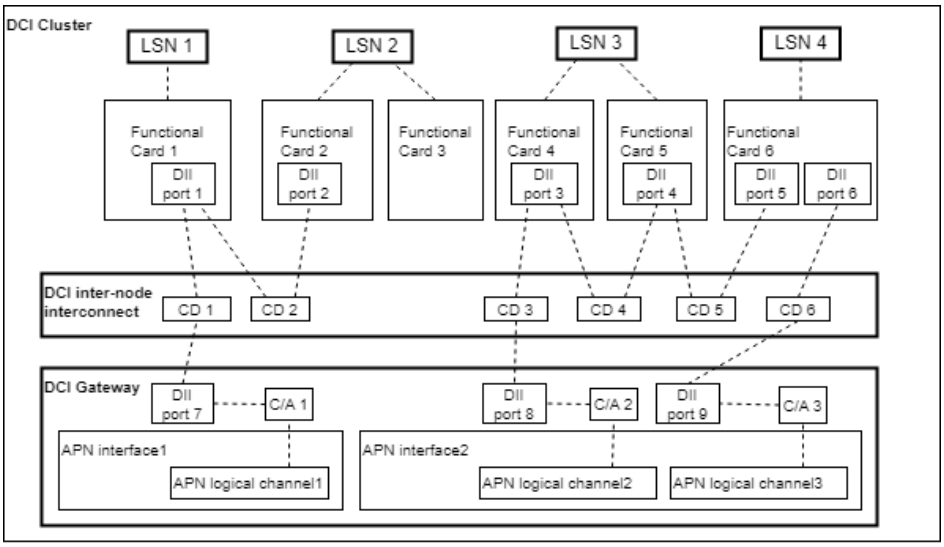
In terms of the IOWN GF DCI Functional Architecture, the patch panel, the IPU, DPU, and NIC functional cards, and any other functional cards added to the switch- or router-like server form the DCI Gateway. The switch- or router-like server(s) that form the DCI Gateway themselves correspond to logical service nodes. Further components of the DCI Functional Architecture in this figure are realized in a manner similar to the one shown in Appendix B.2.3.

Finally, whether an approach as outlined above could actually be feasible would depend on a variety of factors. However, here the intention is to illustrate the degree of flexibility DCI implementors have; the discussion for which use-cases the approach as shown in Figure B-5 is viable, if any, is beyond the scope of this document.

B.3 Example of an instance of the DCI cluster model

This section presents an example of an instance of the basic DCI cluster model proposed in Section 7.3. The DCI Cluster is modeled using UML class diagram notation. By showing instances of the classes, this section aims to provide useful information to interpret the class diagram.

Figure B-6 shows an example of the instances. A box in a box represents a composition relationship. A dashed line represents an aggregation relationship. In this example, there is one DCI Cluster that has four LSNs, one DCI inter-node interconnect and one DCI Gateway.



LSN: Group of functional cards communicating using memory semantics (=LSN)
DII port: DCI Inter-node interconnect port
CD: Connection description (with QoS parameters)
C/A: Connector/adaptor between APN and Inter-Node Interconnect

Figure B-6: Example of an instance of the DCI cluster model

The following items highlight some of the main points of this example:

- LSN 1 is comprised of one Functional Card. LSN 2 is comprised of two Functionals Cards.
- Functional Card 1 has one DCI Inter-Node interconnect (DII) port. Functional Card 6 has two DII ports. Functional Card 3 has no DII port.
- The DCI Inter-node interconnect has six Connection descriptions.
- Connection description (CD) 1 has a relationship to DII port 1 in Functional Card 1 and another relationship to DII port 7 in the DCI Gateway. CD 2 has two relationships to DII ports in Functional Cards.
- The DCI Gateway has two APN interfaces, three DII ports and three Connector/adaptor(C/A).
- C/A 1 has a relationship to APN logical channel 1 and a relationship to DII port 7.

In the basic DCI cluster model in Section 7.3, DCI Physical Node is not modeled because the DCIIO does not need to care about the DCI Physical Nodes. The two Functional Cards of LSN 2 might be in one DCI Physical Node and the two Functional Cards of LSN 3 might be in different DCI Physical Nodes. Such location information is intentionally hidden in the DCI cluster model and in this example.

The intra-node interconnect is not modeled because DCIIO does not need to care about it. In this example, it is assumed that Functional Card 2 and 3 are connected via a hidden intra-node interconnect.

In the future, the basic DCI cluster model will be enhanced taking the future extensions in Section 7.4 into account. This instance example will be modified accordingly.

History

Revision	Release Date	Summary of Changes
1	January 2022	Initial Release
2	April 2023	Version 2