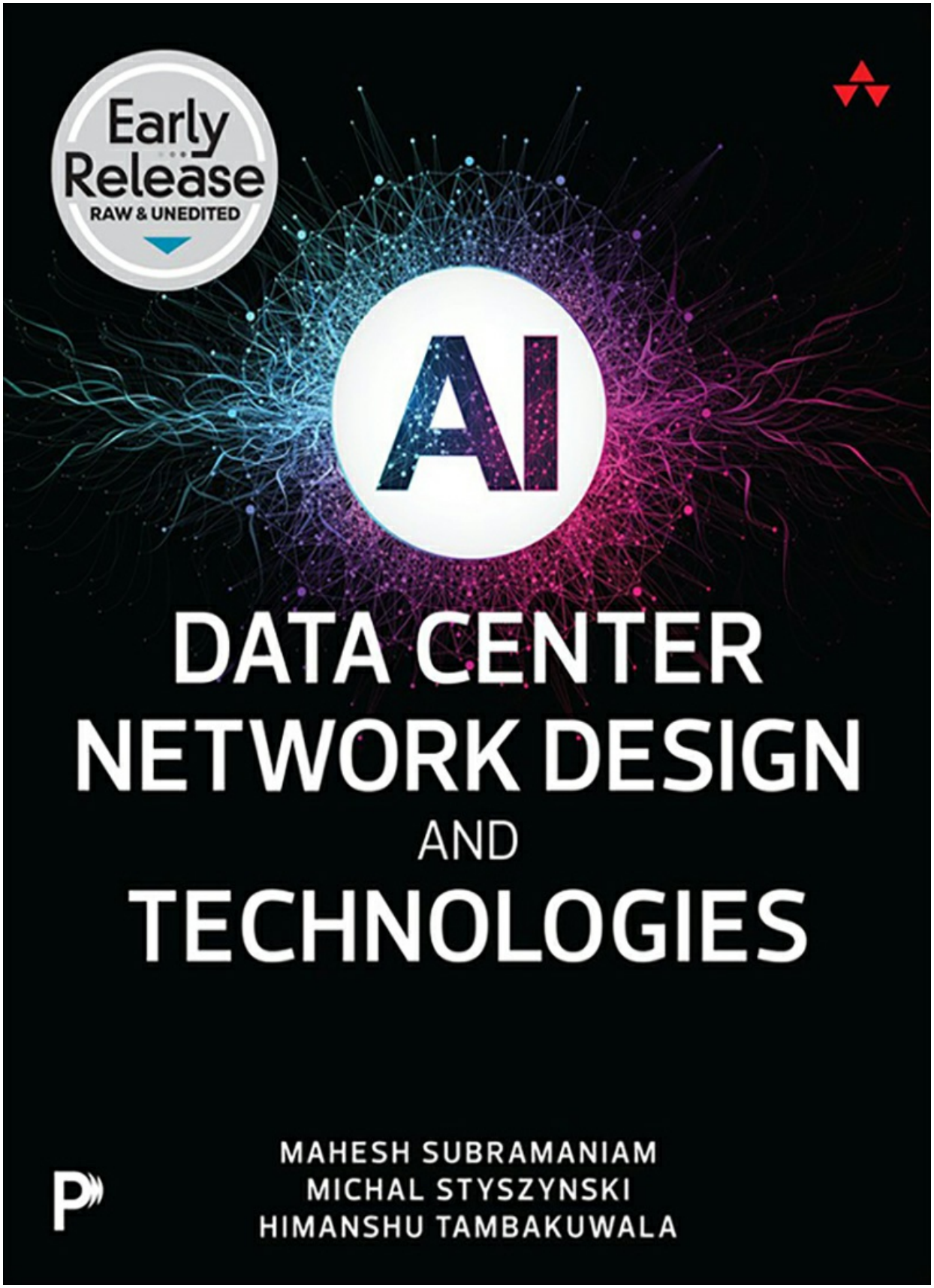
DATA CENTER NETWORK DESIGN AND TECHNOLOGIES

MAHESH SUBRAMANIAM
MICHAL STYSZYNSKI
HIMANSHU TAMBAKUWALA


Early
Release
RAW & UNEDITED
AI
DATA CENTER
NETWORK DESIGN
AND
TECHNOLOGIES
MAHESH SUBRAMANIAM
MICHAL STYSZYNSKI
HIMANSHU TAMBAKUWALA

AI Data Center Network Design and Technologies

AI Data Center Network Design and Technologies

Mahesh Subramaniam, Michal Styszynski, Himanshu Tambakuwala

A NOTE FOR EARLY RELEASE READERS

With Early Release eBooks, you get books in their earliest form—the author's raw and unedited content as they write—so you can take advantage of these technologies long before the official release of these titles.

If you have comments about how we might improve the content and/or examples in this book, or if you notice missing material within this title, please reach out to Pearson at PearsonITAcademics@pearson.com

Addison-Wesley

Hoboken, New Jersey

Visit us on the Web: informit.com/aw

Library of Congress Control Number:

ISBN-13: 978-0-13-543628-8
ISBN-10: 0-13-543628-1

$PrintCode

Head of Enterprise Content and Training, Enterprise Learning and Skills
Julie Phifer
Executive Editor
James Manly
Development Editor
Ellie C. Bru
Managing Editor
Sandra Schroeder
Senior Project Editor
Mandie Frank
Copy Editor
Kitty Wilson
Indexer
Proofreader
Technical Reviewer
Tanveer Dandiana
Designer
Chuti Prasertsith
Compositor
codeMantra

To my brother, Rajesh, who taught me the meaning of connection—long before I ever studied networks. “Your absence echoes in every page, a quiet hum between the lines.…A bright star in my sky, the legacy I live and build for.…You are the architecture of my soul—my reason to dream, my need to create, my home in every connection.”

I also want to dedicate this book to my parents, Sarasu and Subramaniam, my wife, Ramya, and my sons, Rithvik and Rithish. They are my steady roots beneath every life storm, the strength that holds the ground I walk upon. Their love is my unwavering light; that light is the one and only thing which makes me stand.

—Mahesh

I want to dedicate this book to my wife, Kasia, and my sons, Ernest and Marcel, for their love, patience, and understanding. Their smiles, encouragement, and joy for life inspire me every day and have helped me complete this book.

—Michal

To the ones I owe it all: Neha, Rachit, Aanvi, and my parents, Prakash and Indu. Thank you for every late night and every quiet sacrifice you made.

—Himanshu

Contents

Table of Contents

Foreword

This content is currently in development.

Preface

AI Data Center Network Design and Technologies is a comprehensive, practical guide for network engineers, architects, and technology leaders who are building, scaling, or optimizing the infrastructure that supports artificial intelligence. This book bridges the gap between theory and practice, providing an in-depth look at the design, deployment, and management of high-performance AI data centers—where scale-out and scale-up networking, xPU-based computing, storage, and operations work together.

Register your copy of *AI Data Center Network Design and Technologies* on the InformIT site for convenient access to updates and/or corrections as they become available. To start the registration process, go to informit.com/register and log in or create an account. Enter the product ISBN (9780135436288) and click Submit. Look on the Registered Products tab for an Access Bonus Content link next to this product, and follow that link to access any available bonus materials. If you would like to be notified of exclusive offers on new editions and updates, please check the box to receive email from us.

Acknowledgments

We take this opportunity to sincerely acknowledge those individuals who have provided insightful meets and encouragement from the very start of our careers, helping us lay a strong foundation for the advancements in networking technologies we see today:

Manish Prakash, Shivam Srivastava, Raghu Subramanian, Gouthaman Guna, Tim McCarthy, Rajesh Sunkara, Ansuha P, John Steele, Jim Lowe, Ken D, Varun Arya, Rob Nath, Jim Hurd, Mathias Kokot, Sanjeev Ugalkar, Kaushal Agrawal, Karthik Vugane, Vinod Subramaniam (Vinod), Praful Lalchandani, Mike Bushong, Praveen Jain, A.E. Natarajan, Mansour Karam, Murali Vemula, Kiran K.L., Raj Yavatkar, T. Sridhar, Jay Ramalingam, Arun Viswanathan, Jasmeet Sawhney, Yogesh Kumar, Chirag Kachalia, Rajesh Dhople, Dharshan Ravichandran, Ashwin Balaji Sundaresan, Kumuthini Ratnasingham, Francis Guiller, Amit Sanyal, Arun Gandhi, Ben Baker, Himansu Sahu, Mallikarjun Tallapragada, Suryanarayana MNV, Suraj Kumar, Harisankar Ramalingam, Rajashekhar Reddy, Parthipan T.S., Balaji Palanisami, Rajesh Venmanad, Manoj Kulandaivel, Sureshkumar Chinnappan, Pavan Kurapati, Dilip Sundarraj, Murugan Kanniappan, Russ White, Amogh Vijayakumar, Vince Loschiavo, Gilbert Montanez, Gustavo Castellanos, Nek Singh, Kevin Oriet, Vinay Kakkar, Praveen Shetty, Sivakumar Gnanasundram, Shankar Nagaraj, Murali Chandran, Santha Kumar, Shahid Ali Khan, Abey Rajan, and Senthilnathan Ramasubbu.

We also extend our sincere appreciation to:

Nipun Chawala, Rahul Mehta, Aroshi Handa, Rajat, Mike Albano, Ramesh Kandula, Manish Gupta, Tamas Mondal, Kevin Wang, Antoni

Przygienda, Wen Lin, Ajay Gaonkar, Selvakumar Sivaraj, Aquin Mathai, Vrishab Sikand, Kishore Tiruveedhula, Thorbjoern Zieger, Zacharias El Banna, Lars Axeland, Elisabeth Rodrigues, Washid Lootfun, Adrien Desportes, Krzysztof Szarkowicz, Riccardo Belli, Zuhair Makawa, Krishnamachari KR, Ridha Hamidi, Grzegorz Koscian, Piotr Czechowicz, Fathi Ben Zaied, Vikram Nagarajan, Jonathan Scherman, Rene Triana, Danny He, Chris Hackett, Muzamil Khan, Jeff Haas, Andy McCarthy, Pawel Rabiej, Pawel Kocimowski, Ahmed Bilal Khan, Slawomir Karas, and Jeffrey Zhang.

About the Authors

Mahesh Subramaniam is a proven leader in AI data centers and next-generation networking technologies. He played a key role in defining the advanced software roadmap for AI fabrics, which are now deployed in production networks across various AI data centers worldwide. As the Senior Director of Product Management for AI Data Centers at HPE Juniper Networks, he leads cutting-edge innovations in AI infrastructure and cloud-scale solutions, optimized for both scale-up and scale-out architectures. Mahesh is also an inventor with several technology patents and a recognized speaker at global forums, including the UEC Summit, OCP, and Tokyo MPLS forum. His work has earned him accolades, including the CEO Excellence Award, the Record High Business Award, and the Star Award for the Cloud DC Reference Architecture. With a remarkable history in the networking industry, Mahesh has a strong track record of leading products and managing technical and business strategies across cross-functional teams.

Michal Styszynski is a Product Management Director in the Data Center Networks Business Unit (DC BU) at HPE Juniper Networking. Michal has been with Juniper Networks for more than 13 years. Before his current role, he was a Technical Marketing Engineer (TME) in the DC BU and a Technical Solution Consultant at Juniper. In these roles, he handled data center projects for large-scale enterprises and federal networks and worked closely with Tier 2 cloud and telco-cloud service providers. Before joining Juniper, he spent around 10 years working at Orange, FT R&D, and TPSA Polpak engineering. Michal graduated from the Electronics & Telecommunications department at Wroclaw University of Science & Technology with a master’s degree in engineering. He also holds an MBA from Paris Sorbonne Business School and is a JNCIE-DC#523, as well as PEC, PLC, and PMC certified from the Product School in San Francisco.

Himanshu Tambakuwala is a highly accomplished networking expert and certified technical architect whose experience spans the entire product lifecycle—from hands-on engineering to product strategy. He is a JNCIE holder in Data Center and Service Provider technologies and an inventor with four granted technology patents and two additional patents currently filed. As a Product Manager at Juniper Networks, Himanshu was instrumental in defining the feature roadmap for network fabrics that power cutting-edge AI/ML data centers.

What You'll Find Inside

Chapter 1: Wonders in the Workload

Start your journey by examining the complete lifecycle of AI/ML workloads. This chapter explains how raw data is collected, labeled, and preprocessed before being routed into distributed training pipelines. You will understand the fundamentals of forward and backward propagation, gradient descent, and iterative optimization—and see how these processes scale across thousands of GPUs through data, pipeline, and tensor parallelism. The chapter introduces job completion time (JCT) and tail latency as key metrics and discusses how RDMA (in RoCEv2) facilitates the low-latency, high-throughput transfers needed for modern AI. The technical groundwork clarifies why lossless, high-radix, dynamically balanced fabrics are crucial.

Chapter 2: "The Common-Man View" of AI Data Center Fabrics

This chapter examines the different requirements for AI training and inference data centers. It provides a technical comparison of InfiniBand and Ethernet fabrics, emphasizing their trade-offs in latency, scalability, cost, and ecosystem support. The section discusses AI-specific traffic patterns, such as low-entropy flows, elephant bursts, and synchronization-induced congestion, and explains how advanced load-balancing techniques (static, dynamic, and global) and congestion management methods (ECN, PFC, and DCQCN) are used to maintain fabric efficiency and resilience.

Chapter 3: Network Design Considerations

In this chapter, the book becomes a practical blueprint for building scalable AI clusters. It introduces rail-optimized design and rail-unified design (aka non-rail optimized design), explaining how each approach affects latency, scalability, and fault tolerance. You'll get hands-on guidance for rack layouts (ToR, MoR, and EoR), cable management, and power/cooling planning. The text then expands to advanced topologies—Clos, Dragonfly, and Torus—showing how to scale from a few racks to tens of thousands of GPUs and how to adapt these principles for inference-centric data centers.

Chapter 4: Optics and Cables Management

Dive into the physical layer with a technical tour of optics and cabling. This chapter covers the evolution from 10 Gbps to 400 Gbps/800 Gbps/1.6 Tbps, the role of DSPs, advanced modulation (PAM4 and QAM), and FEC. It compares transceiver types (QSFP, OSFP, and CFP), connectors (MTP/MPO and LC), and cable options (DAC, AEC, AOC, MMF, SMF, and DWDM). The chapter also explores the latest in pluggable, co-packaged, and linear-drive optics, emphasizing the importance of power and thermal management in high-density environments. We have note addressed the upcoming CPO and LRO/LPO optics technologies in this book as they are subject to ongoing architectural development and discussions with multiple vendors regarding power-efficient rack solutions.

Chapter 5: Thermal and Power Efficiency Considerations

AI clusters push power and cooling to their limits. This chapter addresses the engineering required to keep them running. You'll learn about airflow management (front-to-back, back-to-front, and bidirectional), advanced liquid cooling (immersion, cold plate, rear-door heat exchangers, and spray cooling), and the impact of high-density servers and optics on rack design. Real-world examples from hyperscalers illustrate how next-generation clusters are redefining what's possible in data center thermal management.

Chapter 6: Efficient Load Balancing

This chapter is the technical heart of the networking discussion. It starts with the limitations of traditional ECMP and hash-based load balancing and then introduces a suite of modern technologies: static load balancing, dynamic load balancing (DLB), global load balancing (GLB), flowlet-based rebalancing, per-packet spraying, and selective spraying for RDMA. You'll see how each method works, where it excels, and how to tune it for AI/ML traffic patterns, ensuring optimal utilization and minimal congestion.

Chapter 7: RoCEv2 Transport and Congestion Management

This chapter provides an in-depth technical overview of RoCEv2, the primary transport protocol for AI clusters. The chapter details every potential congestion point and explains the mechanisms—including ECN, PFC, DCQCN, SFC, and CSIG—that can be used to manage them. You'll learn how to tune these controls for different environments, how to prevent PFC storms, and how to use emerging techniques for even more precise congestion management.

Chapter 8: IP Routing for AI/ML Fabrics

Routing is the silent backbone of AI performance. This chapter explores BGP, OSPF, IS-IS, and RIFT in the context of AI data centers, detailing advanced features such as BGP unnumbered, add-path, bandwidth communities, and deterministic path forwarding. You'll learn how to design for multi-tenancy, overlay networks, and server-level routing, as well as how to integrate telemetry and controllers for adaptive, performance-aware routing.

Chapter 9: Storage Network Design and Technologies

Training is fundamentally an I/O challenge. This chapter reviews storage technologies—block, file, and object storage; NVMe-oF; parallel file systems; and GPUDirect Storage—and explains the protocols, state machines, and design patterns that enable high-performance, scalable, and

resilient storage networks. The chapter provides practical advice on integrating on-premises and cloud storage, as well as on designing for both hot and cold data paths.

Chapter 10: AI Network Performance KPIs

This chapter outlines the key KPIs—throughput, latency, accuracy, power, efficiency, and scalability—and presents the MLCommons/MLPerf benchmarking suite. You'll learn how to create and analyze benchmarks for both training and inference, ensuring that your performance claims are meaningful, reproducible, and actionable.

Chapter 11: Monitoring and Telemetry

Operations is where design meets reality. This chapter covers the tools and techniques for monitoring and telemetry, from SNMP and syslog to streaming telemetry and in-band flow analyzers. You'll see how to combine real-time and historical data, leverage AI-driven analytics, and automate corrective actions to keep your fabric healthy and performant.

Chapter 12: Ultra Ethernet Consortium (UEC)

This chapter introduces the Ultra Ethernet Consortium and its new protocol stack, which is designed for million-node AI clusters. You'll learn about technical innovations in transport (UET), congestion management (NSCC, RCCC, and CBFC), and packet delivery (ROD, RUD, RUDI, and UUD) and see how UEC compares to InfiniBand and RoCEv2. The chapter offers practical guidance for transitioning to UEC-ready fabrics and highlights the challenges and opportunities ahead.

Chapter 13: Scale-Up Systems

The emergence of scale-up systems represents a significant development in AI data center infrastructure. This chapter details advanced integrated high-performance computing platforms for AI data centers, referred to as super-

accelerators. These platforms employ more than eight XPUs (accelerators) within a single system rack, interconnected via next-generation technologies such as UALink, which is a viable alternative to Nvidia NVLink. The chapter presents an overview of encapsulation formats and their key characteristics. It also discusses the placement of these new scale-up systems within the data center network and their integration with scale-out backend solutions, including ESUN (Ethernet for Scale-Up Networking), as proposed at a high level by the Open Compute Project Foundation (OCP).

Part 1: AI/ML Data Center Design Workloads and Requirements

Chapter 1. Wonders in the Workload

What's New in AI Data Center Workloads

Right now, we are in era of generative AI, which means that AI is being used for generating new content based on inputs. The input could be in different formats—like images, conversations, audio, or video—and the system needs to be able to understand the input correctly. Then it needs to generate accurate output. The content being generated can be of any type: text, audio, image, video, or animation.

Different AI models are used to generate different content or different results. These models use algorithms that look at patterns in existing data sets to generate content. Hence, data is crucial for generative AI applications, such as large language models (LLMs), image classification, or protein design. Raw data is collected from various places, like specific database sources, Wikipedia, and even other Internet sites. Data scientists or engineers preprocess this raw data, labeling or tokenizing each piece of data. The tokens need to be trained to produce the desired result.

For example, you might use a large language model to get relevant answers for a query. To do that, you need a model that has been trained on a relevant data set. This training involves feeding huge amounts of data (petabytes) into a training cluster. The more relevant data that is used for training, the more robust the model will be.

The Life Cycle of an AI Model

Figure 1-1 illustrates the different stages of AI workloads, which are detailed in the following list:

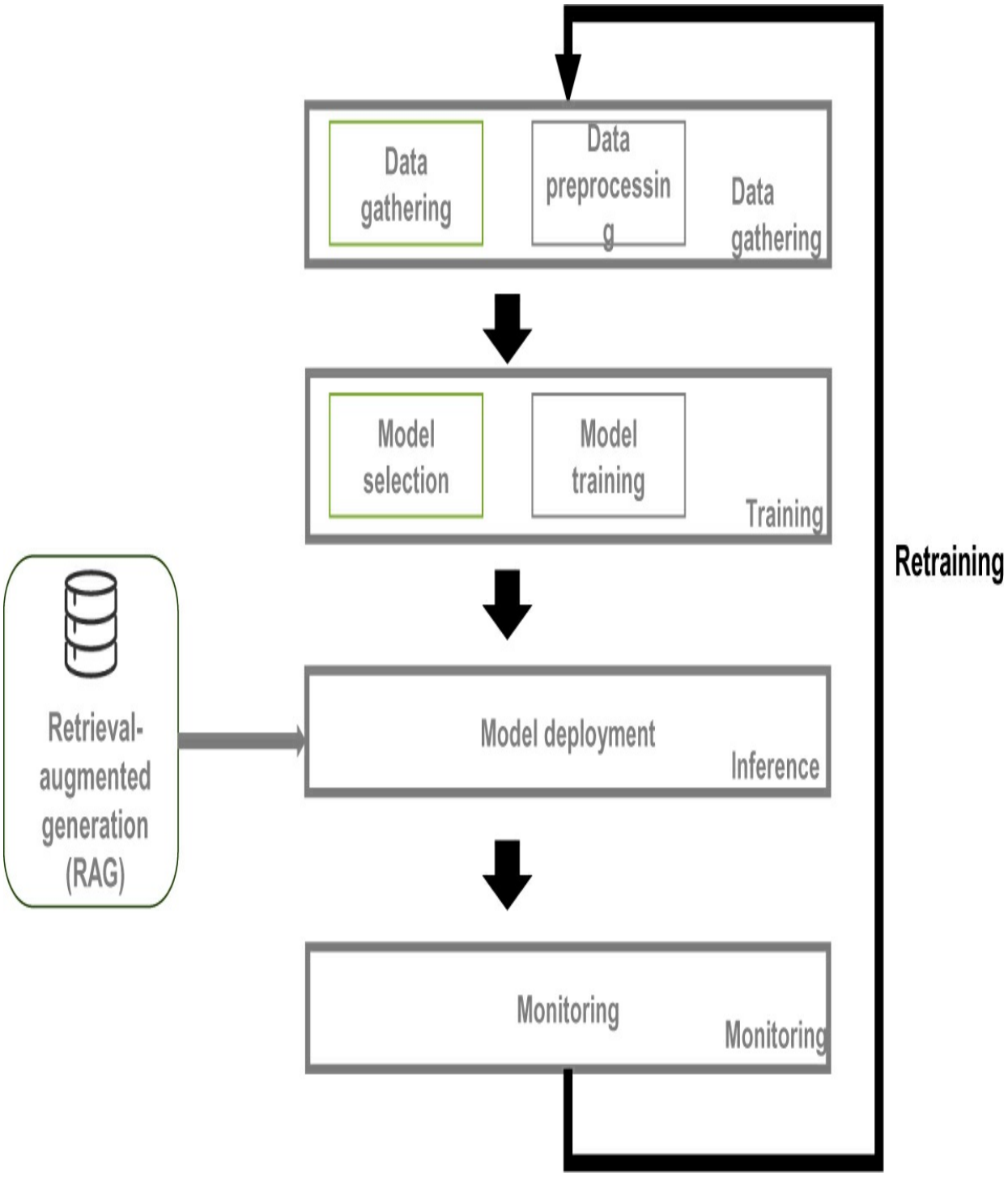


Figure 1-1 *Stages of processing AI/ML workloads*

- **Data gathering:** Data gathering involves collecting data from various sources. Currently, the companies that have the most data have a clear advantage. The quality of data is also important. Using data that is not

relevant to the required use case can impact the accuracy of model. In general, data needs to be processed to remove noise and duplication, and it should be tagged. Tagging makes it possible to get the right data set for the specific business problem the model aims to solve.

- **Training:** Training involves selecting an algorithm, often referred to as a model, and a set of parameters, which are variables to be used by the algorithm. The model is run across various data sets. It is run iteratively on the curated data. The result of the iteration is checked to validate the accuracy based on the existing inputs and corresponding expected output. The results of each iteration show the accuracy of the model, parameters chosen, and the gradient derived, which represents the changes required to the parameters. The parameters are adjusted on each iteration to achieve better accuracy. The model then goes through retraining to get the next set of results. This iterative process continues until the model with the desired accuracy is derived. Training yields the model and parameters that give the most accurate results. The parameters can be updated in different ways:
  - **Supervised:** Intervention is required to update the parameters after each iteration.
  - **Semi-supervised:** Some level of supervision is needed.
  - **Unsupervised:** The system itself update the parameters.
- **Inference:** Inference involves deploying the trained model for real-world use and monetization. The AI applications that we interact with in our day-to-day life—like ChatGPT from OpenAI and Gemini from Google—use inference.
- **Retrieval-augmented generation (RAG)/ Model Context Protocol (MCP) :** Using RAG or MCP with inference enables the model to retrieve the latest up-to-date data from a large database and provide more appropriate answers. After retrieving the necessary recent information, the model generates a response or output based on that new information.
- **Monitoring:** The deployed model is monitored for accuracy and periodically retrained on newer data sets.

Training an AI Model

Training is a critical phase in the AI and machine learning (ML) to make correct predictions or decisions using data. During this stage, the model processes input data, performs calculations, and generates output. In AI/ML Training, Forward propagation is the mechanism by which the model produces an initial prediction. And then, the predicted value is compared with the actual value.

Simple Steps:

1. Input Data: Provide the model with a picture, number, or sentence.
2. Calculations: Each layer multiplies input by a weight, adds a bias, and applies an activation function in sequence.
3. Output: The model returns a prediction, such as “This is a pen” or “The price will be $380””.

This is the first iteration.

In the second iteration, the output is compared with the expected output to check the accuracy. If the accuracy is not found to reach the expected level, changes in the weights, or *gradients*, are suggested. The gradients are then applied to parts of the input. This process is referred as backward propagation. The data is again passed through the function of interest, which investigates the parts of the input and weights and generates output. The iterations continue until the desired level of accuracy is achieved.

In this example, we have discussed the steps involved for one set of data. For a model to be scalable, it needs to be robust enough for many sets of data. The same process needs to be repeated for a very large set of data to fine-tune the parameters based on the use cases. The amount of data can range up to petabytes, which means running this process on a single CPU or GPU core in serialized fashion is practically impossible. This is where parallel processing enters the picture. The idea is to include as many processor cores as possible to perform operations simultaneously. The more cores that are used, the less time it takes to train the model. In general, GPUs are preferred for this processing because they can support higher numbers of cores and are easier and cheaper to scale out. A challenge with parallel processing is the

synchronization of results. To address it, collective communication libraries (xCCLs) are used. xCCLs can be used for multiple cores within a node as well as across nodes.

Table 1-1 describes the various training stages.

Table 1-1 *Training Stages*

Training Stage	Description
Training	The data is broken into multiple batches, and the model is trained using a process called forward propagation.
Forward propagation	The model processes the data through different layers, performing advanced math such as matrix multiplication.
Error calculation	At the end of the forward pass, errors are calculated from the real result that the model should have presented.
Backward propagation	The errors are fed back in a backward propagation through the model, where a lot of calculus is done.
Use of local gradients	Local gradients are calculated and used to tweak or learn from the model's parameters.
Collective communication	Collective communication libraries (xCCLs) enable communication among multiple nodes, which is essential for the data transfer that is required with parallel processing. xCCLs ensure fast and efficient communication among multiple GPUs and networking devices across different machines. Some examples of CCLs are the NVIDIA Collective Communication Library (NCCL) from Nvidia, the ROCm Communication Collectives Library (RCCL) from AMD, Gloo from Meta, and UCC from OpenUCX.
Iteration	All of the stages just described are done iteratively for the whole batch of data to train the model and improve at every iterative step.

Parallelism

The training of an AI/ML model can take anywhere from hours to several weeks or even months, depending on the model's size and the data sets being trained. When the data set used for training is very large, the training time required is also high. As we have discussed, it is possible to speed up the training process by dividing the data and processing it on multiple GPUs simultaneously. This process, known as parallelism, involves breaking the data of the model into batches and sharing those batches with multiple GPUs to speed up the training (see Figure 1-2). All the GPUs synchronize their respective batch training results (gradients) using various parallelism techniques, all of which aim to reduce the communication overhead and speed up the training.

To reduce the training time, parallel processing has become the de facto standard. Parallelism can be achieved by using various techniques, including data parallelism and model parallelism (which includes both pipeline parallelism and tensor parallelism). These techniques can be combined to achieve maximum parallelization and speed up the training process.

Data Parallelism

Data parallelism can significantly speed up the training process for AI models. With data parallelism, all GPUs run the same model across different data sets. The results of one GPU need to be shared with all the GPU. So, all GPUs store the results of all the other GPUs. There are three methods of sharing results: *all-reduce*, *all-gather*, and *all-to-all*.

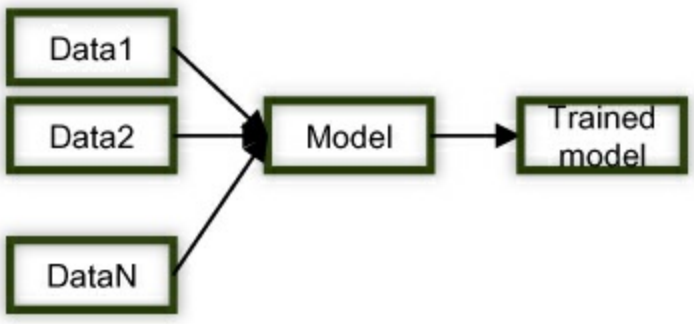


Figure 1-2 *Data parallelism*

Model Parallelism

Model parallelism, illustrated in Figure 1-3, is a technique in which a model is broken down and processed on multiple GPUs. It is used when a model is large and cannot be executed using a single GPU. Model parallelism involves the ring-reduce operation, in which all GPUs run the same data sets across different model layers. The result of the first GPU running the first layer of the model is passed to the second GPU, where it is coupled with the result of the second layer of model. This combined result is then passed on to the third GPU, and so on.

Model parallelism can be further classified into two types of parallelism: pipeline parallelism and tensor parallelism.

With pipeline parallelism, a neural network or model is broken into layers, and each layer is run on a different GPU or subset of GPUs.

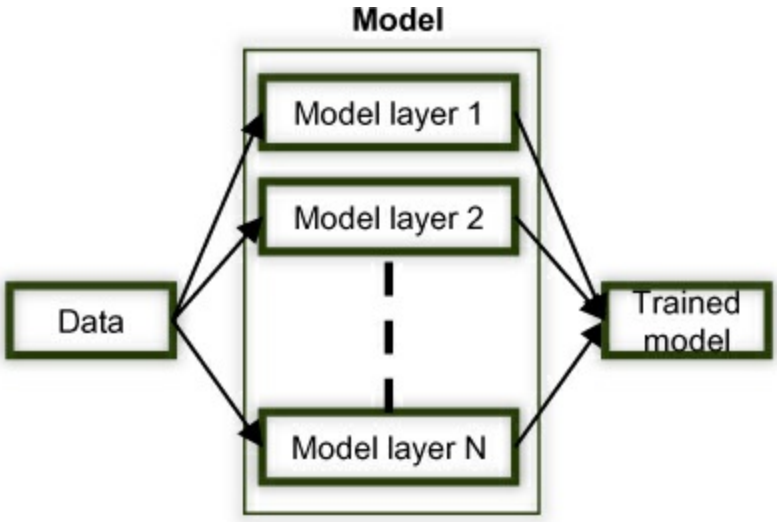


Figure 1-3 *Model parallelism*

With tensor parallelism, operations such as matrix multiplication are broken down and distributed across multiple GPUs within a single server. The layers within the model are further broken into smaller chunks. This process of allowing the operations to be processed simultaneously on different GPUs can significantly speed up the training process for an AI model.

Job Completion Time (JCT)

Job completion time (JCT) is a common term for training clusters in ML fabric. But what does it mean?

When we're talking about JCT, a *job* is an algorithm that helps train a large ML model to get a particular result. Usually, a job is split into multiple tasks, threads are created for each task, and the threads run across different GPUs in parallel to get the result faster. Each thread takes some time to run; this is called the *thread execution time*. Also, each thread needs to communicate its results with other GPUs through the fabric. So, inter-GPU communication is essential to get results.

JCT is a metric that measures the amount of time between the start and end of a job. For example, say that a job is divided into 100 threads, as illustrated in Figure 1-4. Out of 100 threads, 99 threads finish on time, but 1 thread takes a long time at the end due to fabric congestion or tail latency. This delay affects the whole job completion process, causing a delay in getting the result and moving to the next phase of the job. Therefore, overall system performance is directly proportional to the worst-performing thread.

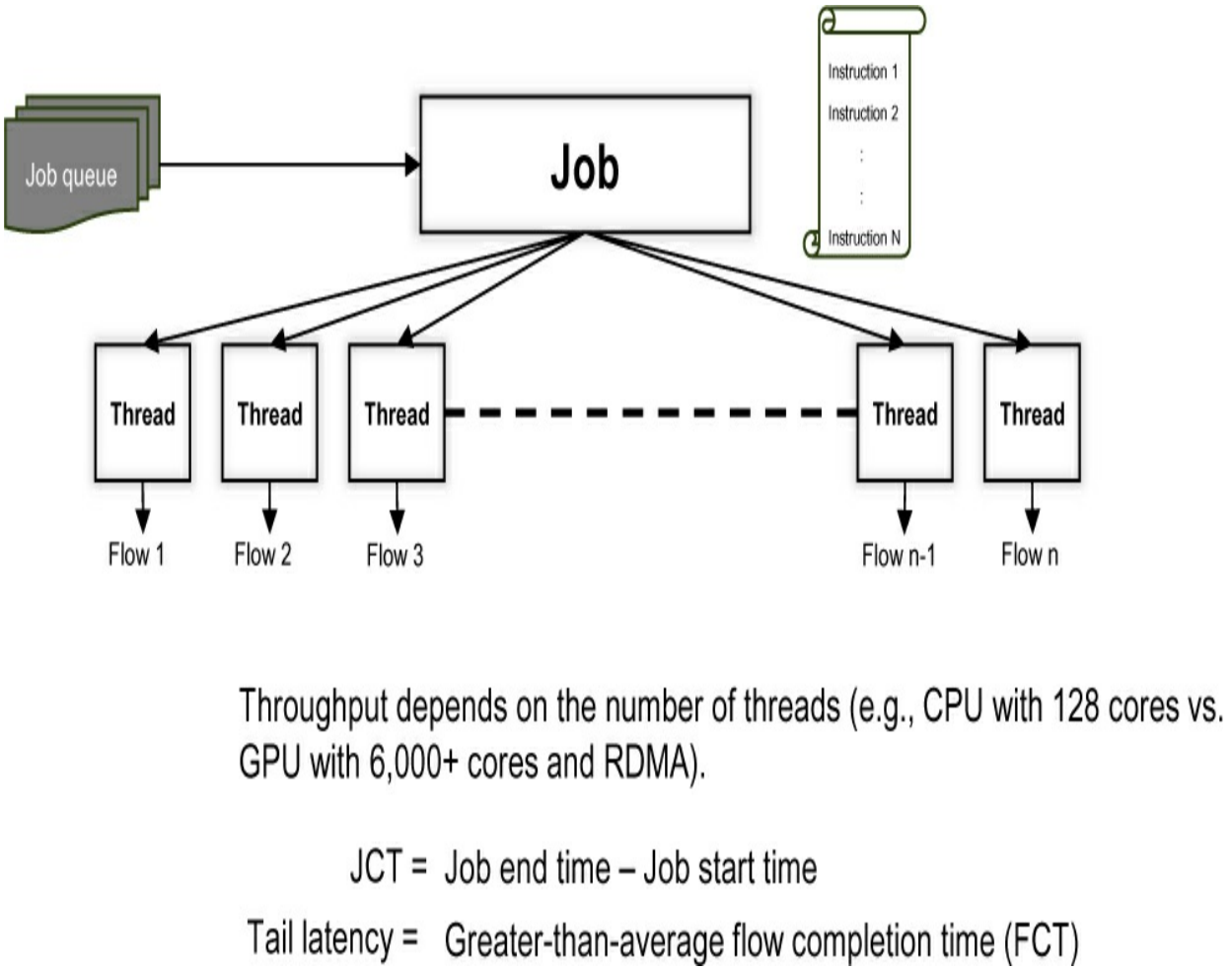


Figure 1-4 *Job completion time*

JCT is a critical KPI in AI/ML networks. To achieve a low JCT, a data center must have a fabric that is free of congestion, that efficiently uses load balancing, and that has low tail latency.

So, ultra-high-throughput switches with high-radix ports are required for AI/ML networks. But with those switches, intensive computation needs to be completed on time, which is the job completion time. To complete a job within a given time, the tail latency needs to be reduced on the fabric. To reduce the tail latency on the fabric, the high-radix switches must have powerful software features, such as dynamic load balancing and optimized lossless fabric using RDMA over Converged Ethernet version 2 (RoCEv2). (In the following chapters, we will talk more about the hardware and software involved and the basic definition of job completion time.)

Tail Latency

Tail latency is high-percentile latency—that is, high latency with a response time longer than the majority of all requests handled by a service or application. For example, Out of 100 process, if just 1% of processing requests observe latency in seconds, and remaining 99% of the processing requests are less than 100 ms, the tail latency will be calculated as seconds.

The impact of tail latency is very critical in serial and parallel job tasks. For example, Figure 1-5 illustrates a situation with *n* tasks of a job, running serially. All the tasks are performed in 1 ms, except for one of the tasks, taskX, which is executed in 1 second. So, the overall job completion time is 1 second + (*n* x 1ms). For example, n=10, with 10 tasks, the result would be 1.010 seconds. The JCT increased a lot in this case due to the high latency of a single job.

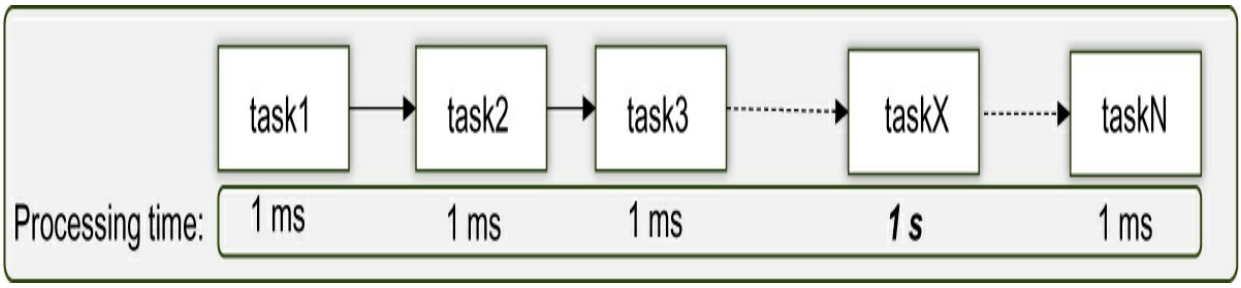


Figure 1-5 *Tail latency with task serialization*

Figure 1-6 illustrates that *n* tasks of a job are running in parallel. All the tasks are performed in 1 ms, except for one of them, taskX, which is executed in 1 second. So, the job has to wait 999 ms for this thread to complete while the rest of the tasks are completed. The overall job completion time is 1 second. The JCT increased a lot due to the high latency of a single job.

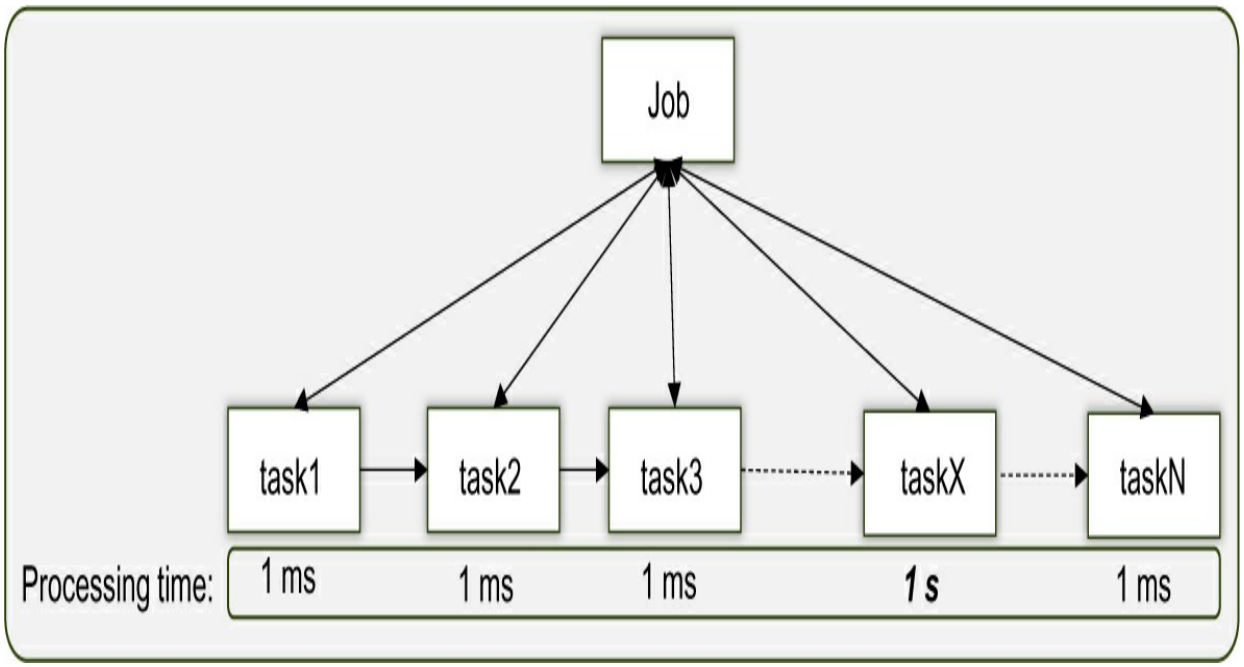


Figure 1-6 *Tail latency with task parallelization*

Think of it in this way: We build a system that performs badly 1 in a million times. It is definitely a very good system. Now, let's say we have to perform a job that has a million tasks to be performed. This means the system is going to perform badly for the job every time. This is the case with AI/ML data centers; they need to perform a very high number of tasks, and tail latency can bring down the overall performance.

In an AI/ML data center, tail latency can impact both the backend, or training clusters, as well as the frontend, or inference clusters. In training data centers, tail latency impacts the overall job completion times, whereas in inference clusters, it impacts the user experience in specific cases.

Understanding Traffic and RDMA

AI/ML workloads use remote direct memory access (RDMA) for data transfer, bypassing the CPU to make the data transfer faster. With RDMA, a network interface card (NIC) directly accesses the remote hosts' memory, bypassing the CPU and kernel. Consequently, an application running on a GPU can directly access memory on a remote host. The many benefits of RDMA include the following:

- Low CPU utilization
- High throughput
- Lower latency due to limited data copy requirements between application memory and data buffers
- Synchronous operations as multiple copies are transferred in parallel with the use of multiple GPU cores

RDMA is a popular data transfer technology for AI/ML training workloads that perform parallel processing in GPUs across multiple servers, resulting in a lot of data transfer.

Figure 1-7 illustrates how the RDMA transfers bypass the kernel networking stack on both hosts. It reduces data copy operations and CPU or kernel bottleneck issues.

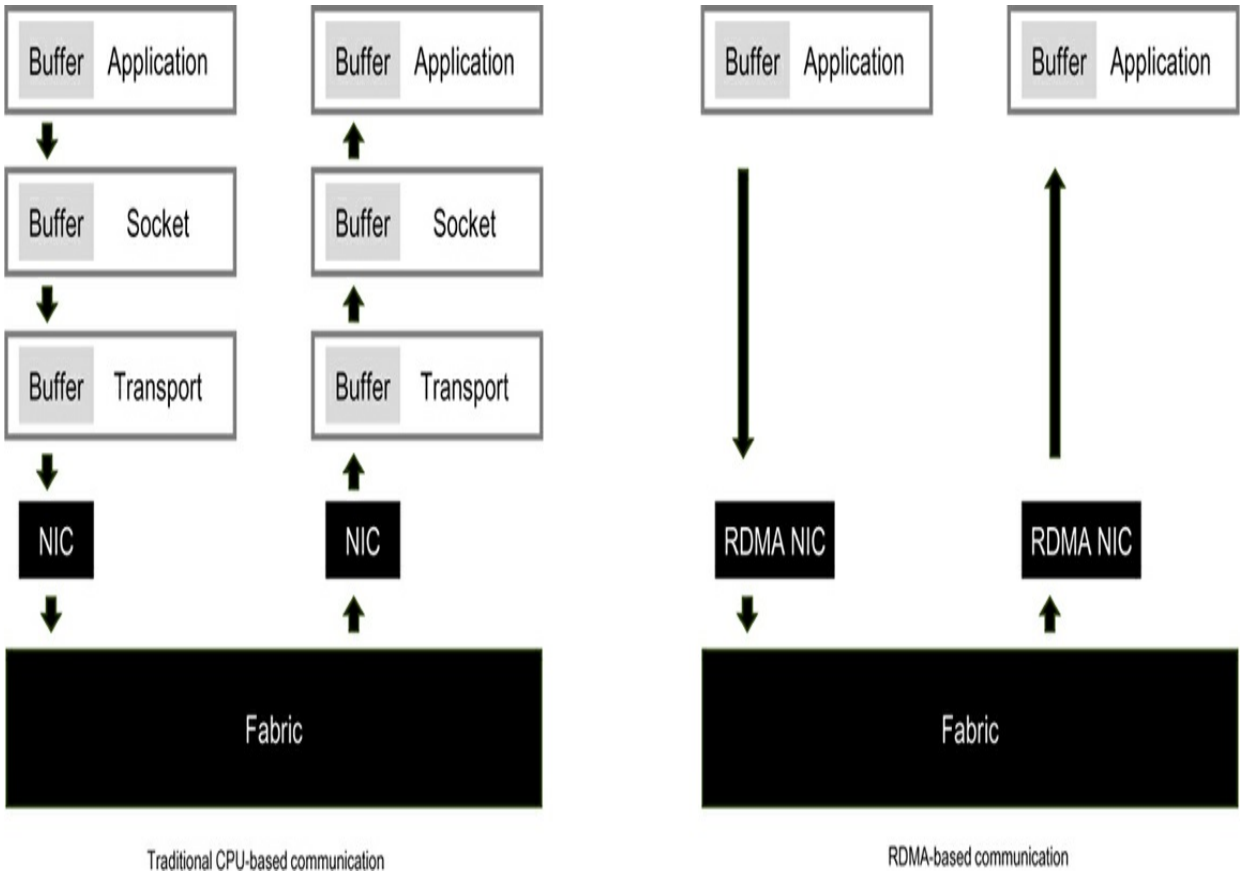


Figure 1-7 *CPU-based versus RDMA-based communication*

With RDMA, an application issues a job using a work queue element (WQE)

or a work request. A WQE is a struct with a pointer to a buffer. The RDMA driver manages the following three queues:

- **Send queue:** Used for data to be sent to a remote host
- **Receive queue:** Used for data received from a remote host
- **Completion queue:** Used to put the WQE in the completion queue once the work request is complete

RDMA supports two types of data transfer:

- **Unidirectional:** For RDMA read, write, and atomic operations
- **Bidirectional:** For RDMA send and receive operations

Figure 1-8 illustrates the RDMA layers within a node. In this case, an application that requires RDMA transfer posts the work requests to the queue. The Verbs API services the application requests. The RDMA driver manages the work queues and address translation, and the RDMA NIC handles transport and reliability of the frames.

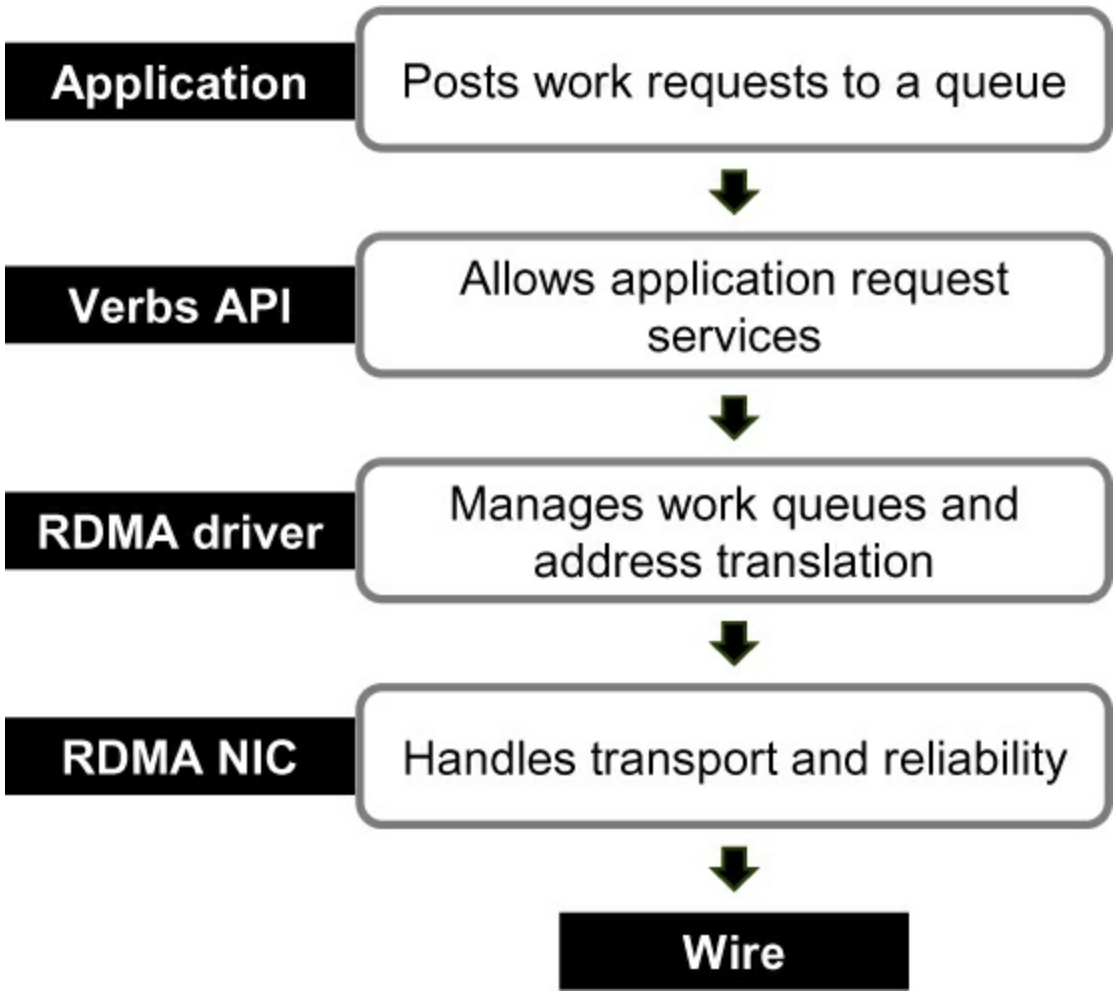


Figure 1-8 *RDMA layers within a node*

RDMA Transport Types

When RDMA must be transported over a network, it needs a transport mechanism. There are three networking protocols that can be used as transport for RDMA:

- **InfiniBand (IB):** IB is governed by the InfiniBand Trade Association (IBTA).
- **RDMA over Converged Ethernet (RoCE):** RoCE is also governed by IBTA. There are two versions of RoCE. The first version introduced Ethernet as the link layer and kept the IB network layer and payload. The second version, RoCEv2, replaced the IB network layer and preserved the InfiniBand Base Transport Header (IB BTH) header and

IB Payload header, with IP and UDP as the outer headers.

- **Internet Wide Area RDMA Protocol (iWARP):** This is a competing solution to IB but not as popular.

Figure 1-9 illustrates the differences between InfiniBand, RoCE, and RoCEv2.

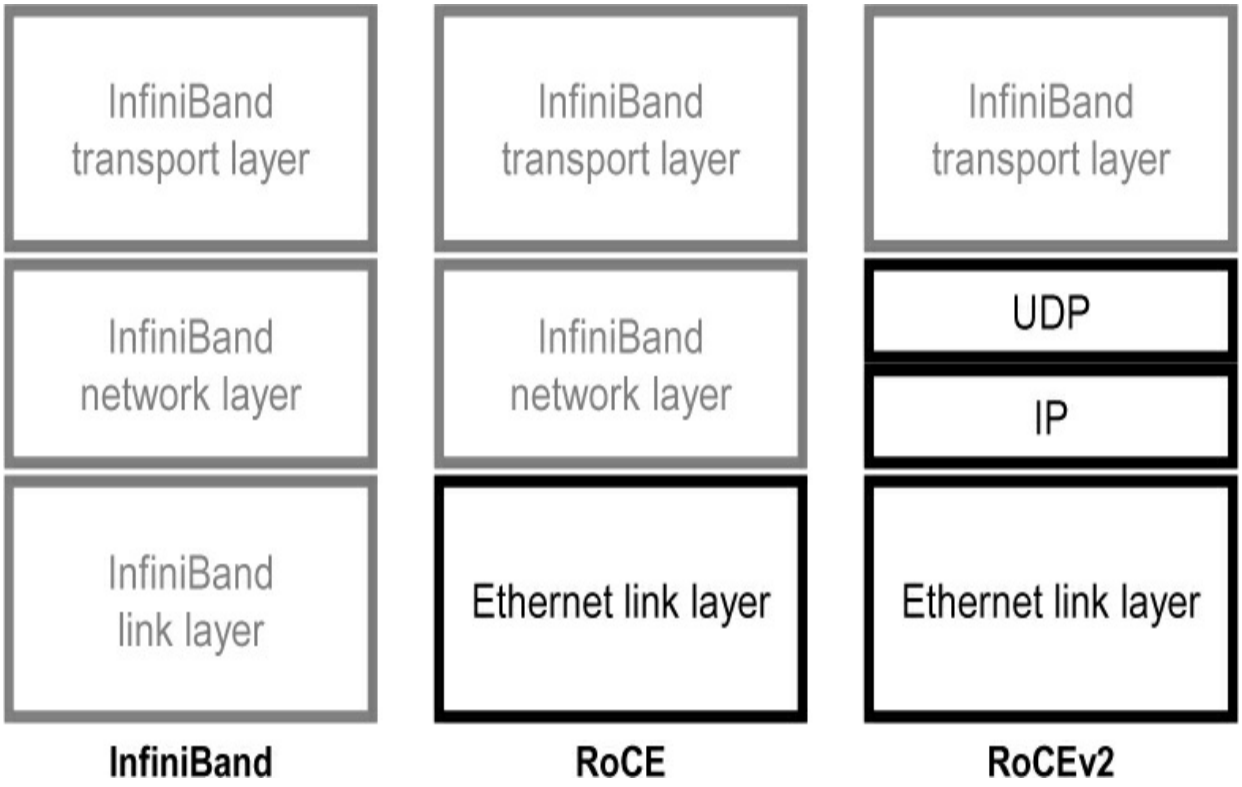


Figure 1-9 *Differences between InfiniBand, RoCE, and RoCEv2*

Figure 1-10 illustrates the differences in the headers for the transport options for RDMA traffic. InfiniBand is a channel-based transport that enables faster communications between endpoints. Ethernet is a traditional standards-based technology for connecting endpoints.

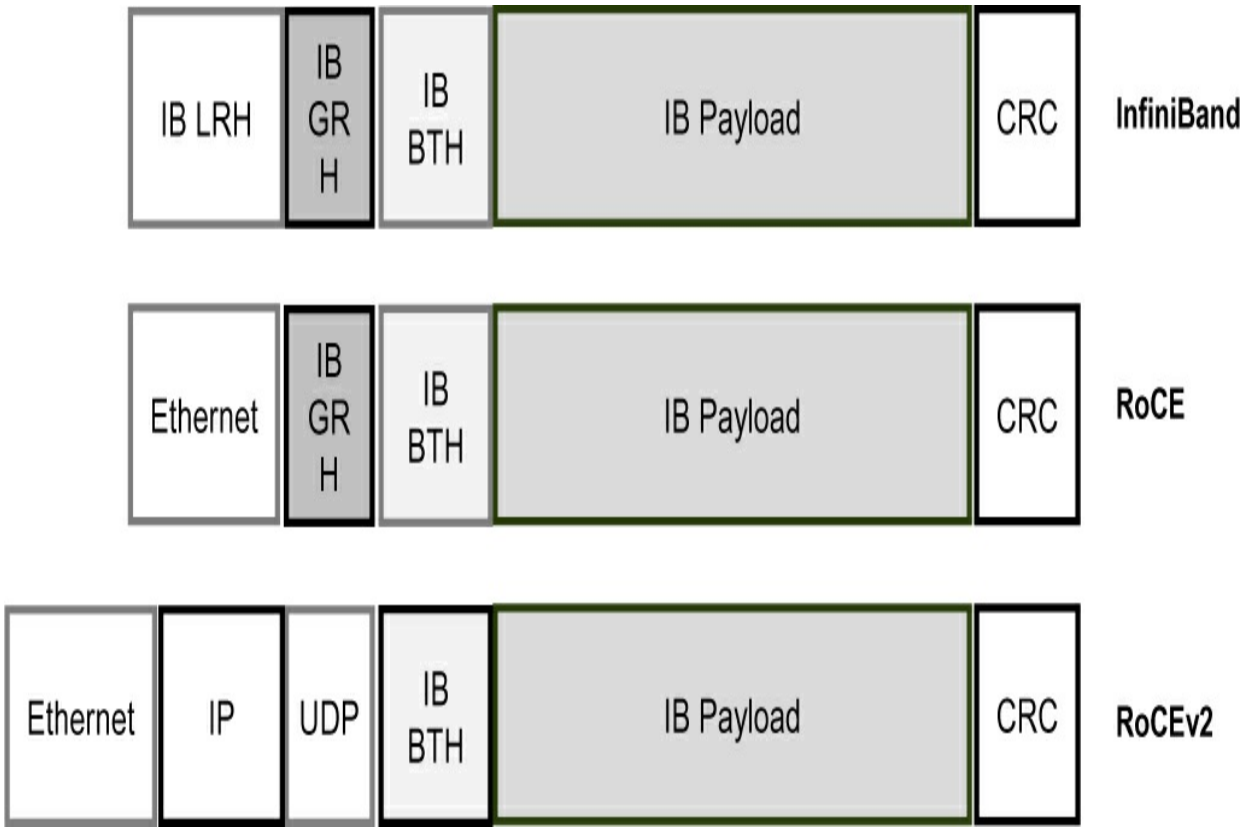


Figure 1-10 *Transport headers for InfiniBand, RoCE, and RoCEv2*

RoCEv2

The RoCEv2 header has IP or IPv6 as the Layer 3 protocol and UDP as the Layer 4 protocol. The UDP destination port is set to 4791, and InfiniBand BTH is the next header. The UDP source port is derived based on the queue pair (QP), which is a combination of send and receive queue ports. Figure 1-11 illustrates the RoCEv2 header.

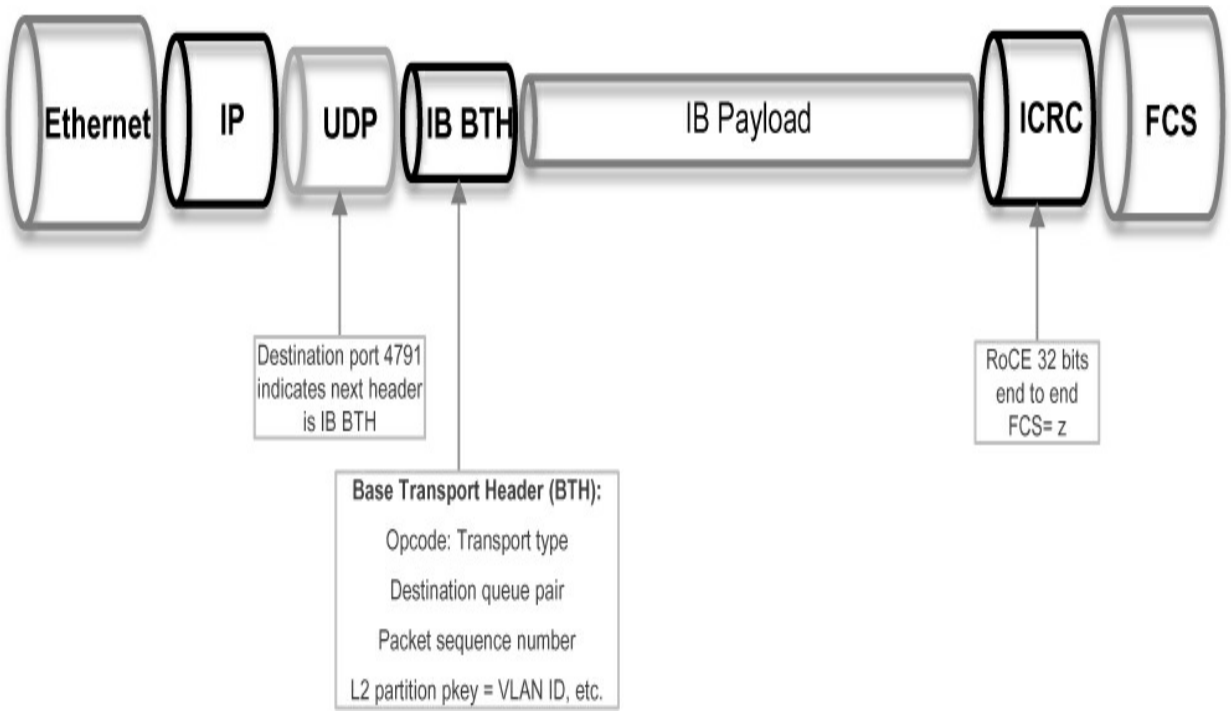


Figure 1-11 *RoCEv2 header*

IB BTH has the opcode to indicate RDMA operations like read, write, send, receive, and more. It also carries the QP information of the destination. Because RoCEv2 uses UDP as the transport, RDMA uses the packet sequence number for reliability. The packet sequence number tracks the delivery sequence of packets.

Figures 1-12 and 1-13 illustrate how a session is formed using RDMA message exchange for read and write operations. First, a three-way handshake is initiated, during which the QP information and sequence number are exchanged between both the server and the client. After the session is established, memory-related information is exchanged based on the type of communication. Subsequently, the data is exchanged. When the data exchange is complete, the session is terminated.

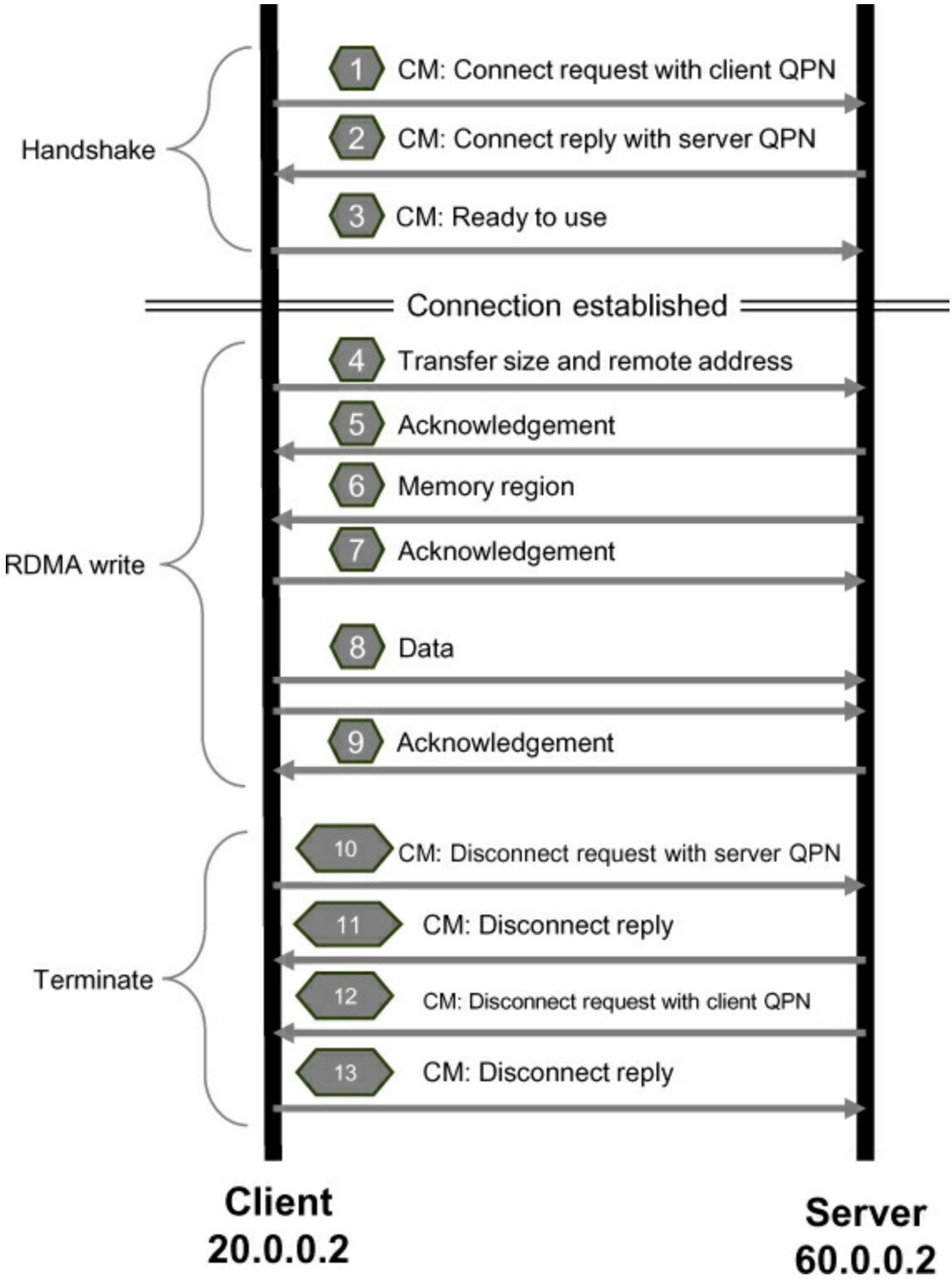


Figure 1-12 *RDMA write message exchange*

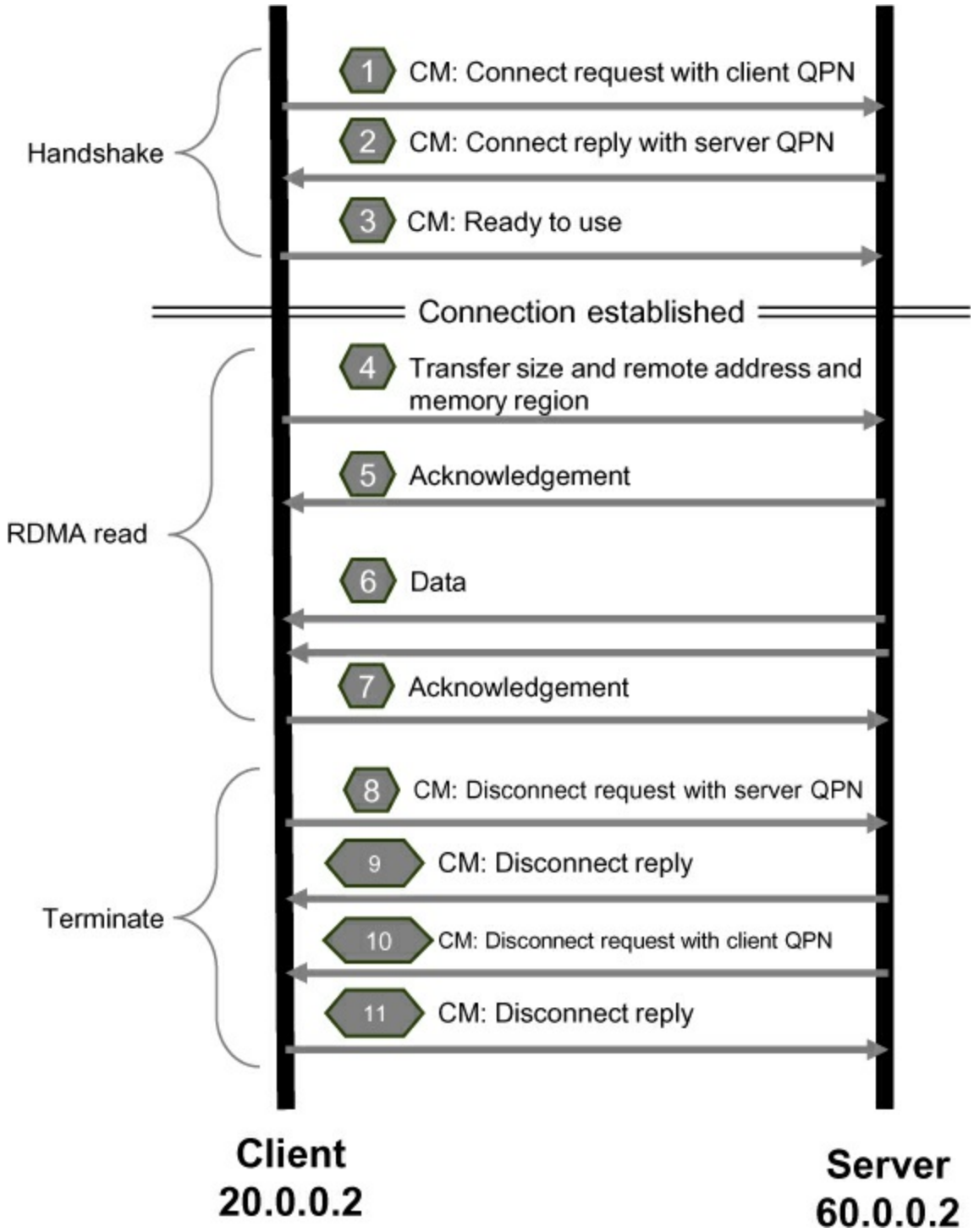


Figure 1-13 *RDMA read message exchange*

The RDMA read request message contains the remote address, key, and size information. The RDMA read response message includes data from the server. Also, notice the acknowledgement request flag in the message, which

indicates that the recipient has received the request. This flag enables selective acknowledgement. Figures 1-14 and 1-15 show the RDMA read request and response packets.

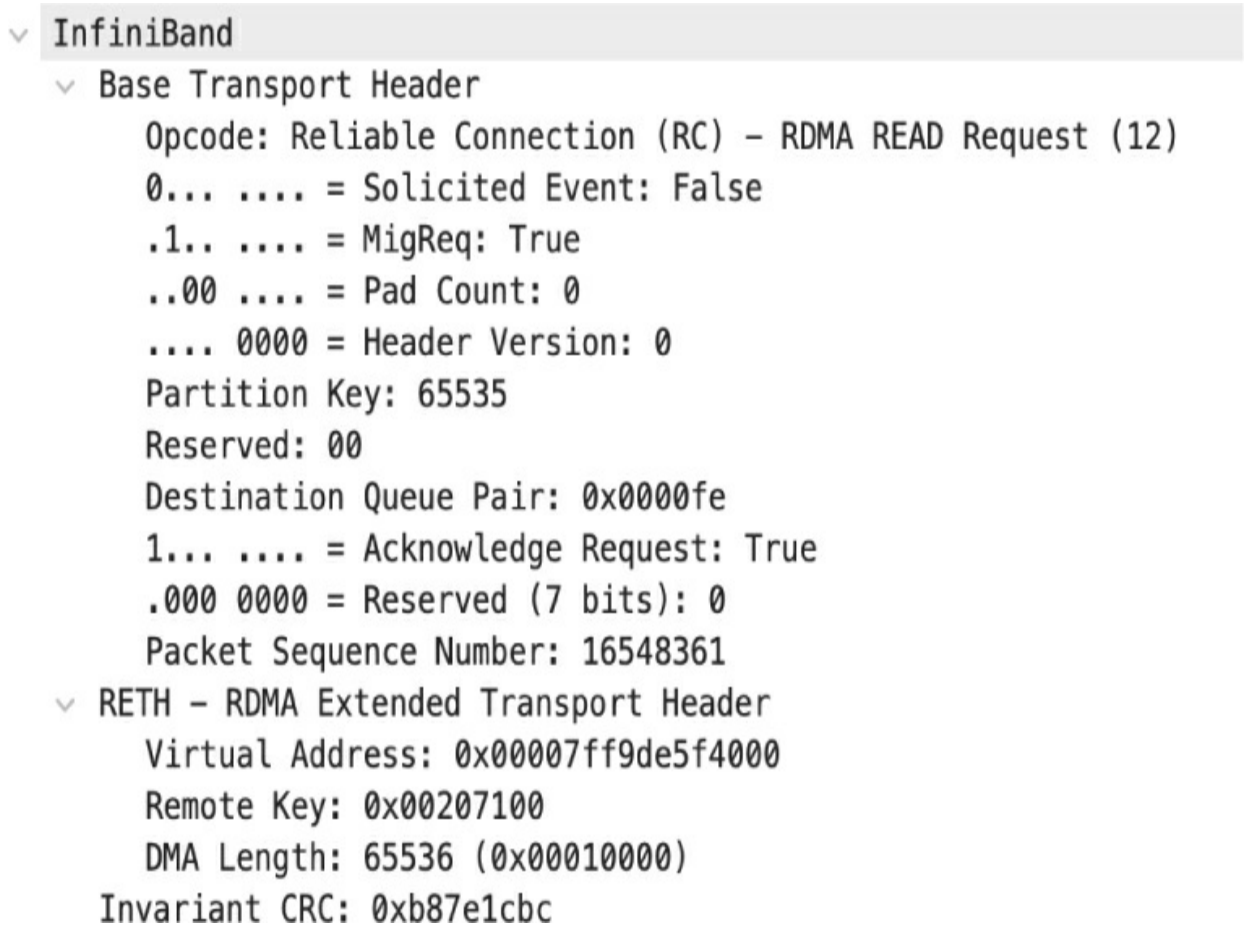


Figure 1-14 *RDMA read request*

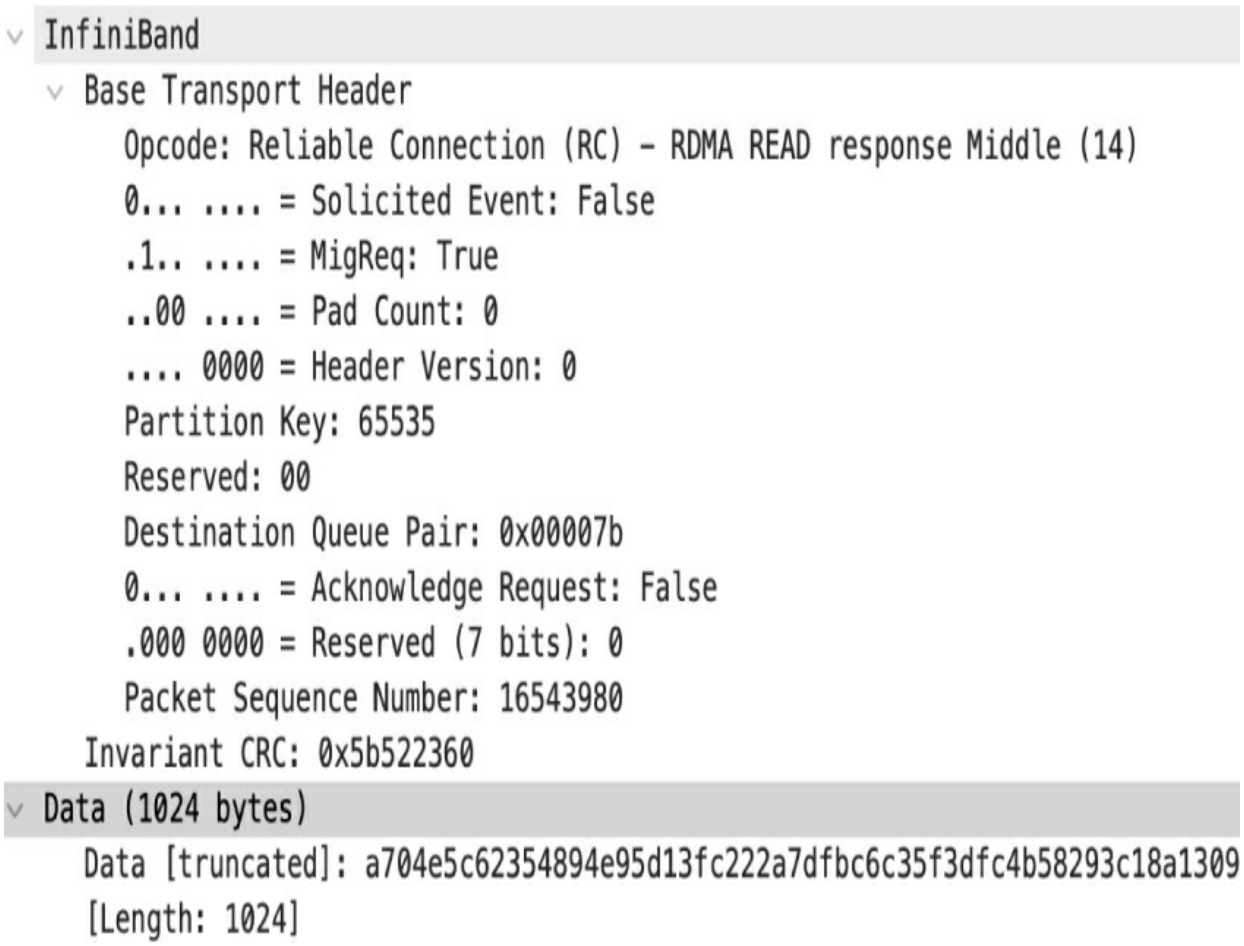


Figure 1-15 *RDMA read response*

Figure 1-16 shows the acknowledgement packet that contains the IB BTH header with the destination QP information as well as the packet sequence number for which it is sending acknowledgement. Above the IB BTH header is the Acknowledge Extended Transport Header (AETH) header, which contains the info to indicate whether it is an acknowledgement (ACK) packet or a not acknowledgement (NACK) packet. The message sequence number (MSN) in the AETH header contains the acknowledgement sequence number of the last message completed at the receiver. This helps in detecting any missed acknowledgment.

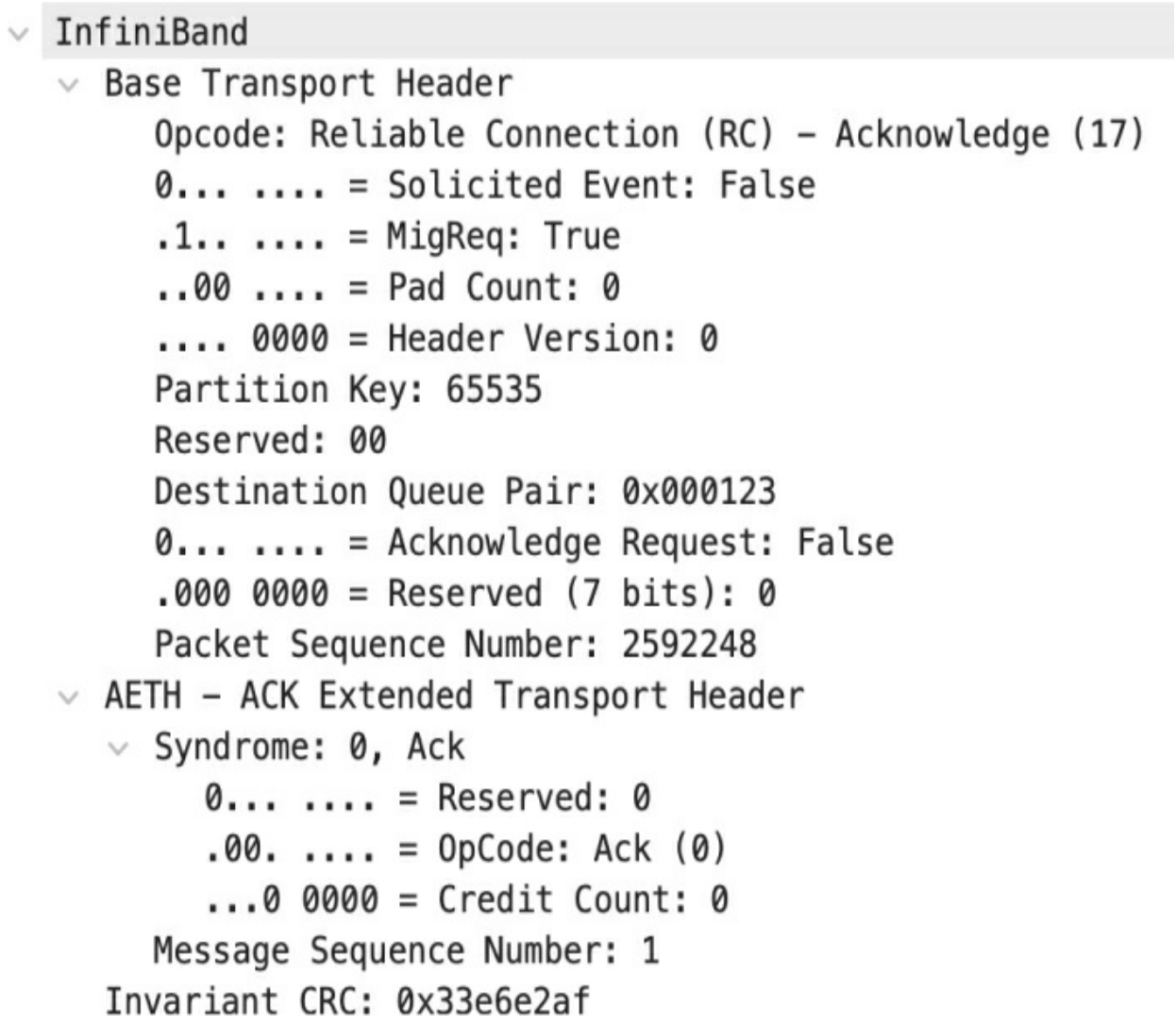


Figure 1-16 *Acknowledgement packet*

Table 1-2 shows the differences between Ethernet and InfiniBand.

Table 1-2 *InfiniBand and Ethernet Comparison*

Characteristic	InfiniBand	Ethernet
Transmission speed	IB supports SDR (Single Data Rate), DDR (Double Data Rate), QDR (Quad Data Rate), FDR (Fourteen Data Rate), EDR (Enhanced Data Rate), HDR (Hundred Gigabit Data Rate), and NDR (Next Data Rate). Slower than Ethernet.	Ethernet supports 10 Mbps, 100 Mbps, 1 Gbps, 10 Gbps, 25 Gbps, 100 Gbps, 200 Gbps, 400 Gbps, 800 Gbps, and 1.6 Tbps
Latency	Lower latency	Higher latency
Reliability	Greater reliable as it is designed for lossless	Inherently a lossy protocol. TCP is reliable, and DCQCN adds reliability to UDP.
Power consumption	Low power consumption	Comparatively higher power consumption
Scalability	Limitations beyond a certain cluster size	Highly scalable
Standardization	Controlled by IBTA and requires vendor lock-in	Open standards enable multi-vendor networks
Cost	Higher cost and lead times	Lower cost and lead times
Versatility	Suitable for certain types of fabrics, like storage, HPC, and AI/ML	Most widely deployed technology and can run in any environment if appropriate protocols are enabled

Summary

As you have seen in this chapter, there are many nuances in an AI/ML workload, mainly on the training side. This is why we titled this chapter "Wonders in the Workload." Novel ingredients—from the training model to inference, job completion time, parallelism, tail latency, and RDMA transport types, including RoCEv2—are interconnected. To get these ingredients to work together, AI data center fabric for GPU cluster interconnectivity is essential. The critical point is that the data center fabric needs to be lossless; it requires high-radix switches with powerful software features, such as

dynamic load balancing, selective routing, and RoCEv2. The next chapter discusses key requirements in AI data center fabrics.

Test Your Knowledge

Chapter Review

The following questions are designed to test your understanding of the content covered in Chapter 1. Following the questions, answers are provided so you can verify your conclusions.

Questions

1. How does the data life cycle influence AI model performance in data centers?
2. Explain the iterative process of training an AI model, including the roles of forward and backward propagation.
3. What are the main types of parallelism in AI training, and how do they address scalability challenges?
4. Define job completion time (JCT) and discuss its significance in AI/ML clusters.
5. How does tail latency affect distributed AI training, and what architectural features help mitigate it?
6. Describe the function and benefits of RDMA in AI/ML data center networks.
7. Compare InfiniBand, RoCEv2, and iWARP as RDMA transport protocols for AI/ML workloads.
8. What are the key requirements for AI data center fabrics to support efficient AI/ML workloads?

Answers

1. The data lifecycle—spanning collection, cleaning, labeling, and

tokenization—directly impacts model accuracy and robustness. High-quality, relevant, and well-labeled data enables models to learn meaningful patterns, while poor data can introduce bias or reduce generalization. The robustness of a model is proportional to the diversity and volume of the training data, which is why petabyte-scale data sets are often used in modern AI/ML workloads.

2. Training involves feeding input data through the model (forward propagation) to generate predictions, comparing these predictions to expected outputs, and then adjusting model parameters using gradients (backward propagation). This process is repeated over many iterations (epochs) and across large batches of data, gradually improving model accuracy.

3. Data parallelism involves splitting data across multiple GPUs, each running the same model; model parallelism involves splitting the model itself across GPUs; pipeline parallelism involves dividing model layers into stages that are processed in sequence; and tensor parallelism involves splitting tensor operations. These approaches enable simultaneous computation, reducing training time and overcoming the limitations of single-processor systems.

4. JCT is the elapsed time from the start of a training job to the end of the job. It is a critical KPI because AI/ML workloads are often split into many parallel tasks, and the slowest task (due to tail latency or congestion) determines the overall JCT. Optimizing JCT is essential for efficient resource utilization and faster model iteration.

5. Tail latency refers to the slowest responses among many parallel tasks. In distributed AI training, high tail latency can delay job completion, as all tasks must finish before the job can proceed. Architectural features such as use of high-radix switches, dynamic load balancing, and lossless fabrics (for example, RoCEv2) help reduce tail latency by minimizing congestion and ensuring even traffic distribution.

6. RDMA (remote direct memory access) enables direct memory access between servers, bypassing the CPU and kernel, which reduces latency, increases throughput, and lowers CPU utilization. It is especially beneficial for AI/ML workloads that require frequent, high-volume data transfers between GPUs across nodes.

7. InfiniBand offers low latency and lossless transport but is less scalable and more expensive. RoCEv2 runs RDMA over Ethernet using UDP/IP, providing scalability and compatibility with existing Ethernet infrastructure, but it requires careful congestion management. iWARP is less popular, offering RDMA over TCP/IP, but with higher latency and less adoption in AI/ML clusters.
8. AI data center fabrics must be lossless, support high throughput and low latency, provide dynamic load balancing, and be capable of handling large-scale parallelism. Features like high-radix 400 Gbps/800 Gbps Ethernet switches, RoCEv2 support, and advanced congestion management—such as DCQCN, including PFC and ECN—are essential for optimally maximizing performance in the AI data center. The design should allow for the integration of a higher number of servers with GPUs; it should help to move, for example, from a three-stage topology to a five-stage topology when the number of AI workloads is increasing over time.

Chapter 2. “The Common-Man View” of AI Data Center Fabrics [This content is currently in development.]

This content is currently in development.

Part 2: AI/ML Data Center Design Concepts

Chapter 3. Network Design Considerations

Background Introduction

As we discussed in Chapter 1, "Wonders in the Workload," processing AI/ML workloads involves three stages: data gathering and preprocessing, model selection and training, and deployment and monitoring.

Figure 3-1 illustrates the different stages of processing AI/ML workloads. Initially, data is gathered from various sources. In inference, the quality of data is directly proportional to the quality of the results. (You have probably seen instances in social media where AI model results are way off from the expectation—sometimes in a humorous way.) In the preprocessing phase, any data that contains errors, missing values, or inconsistencies is either corrected or removed. The data is then identified and tagged with labels. Now the data is ready to be used for model training.

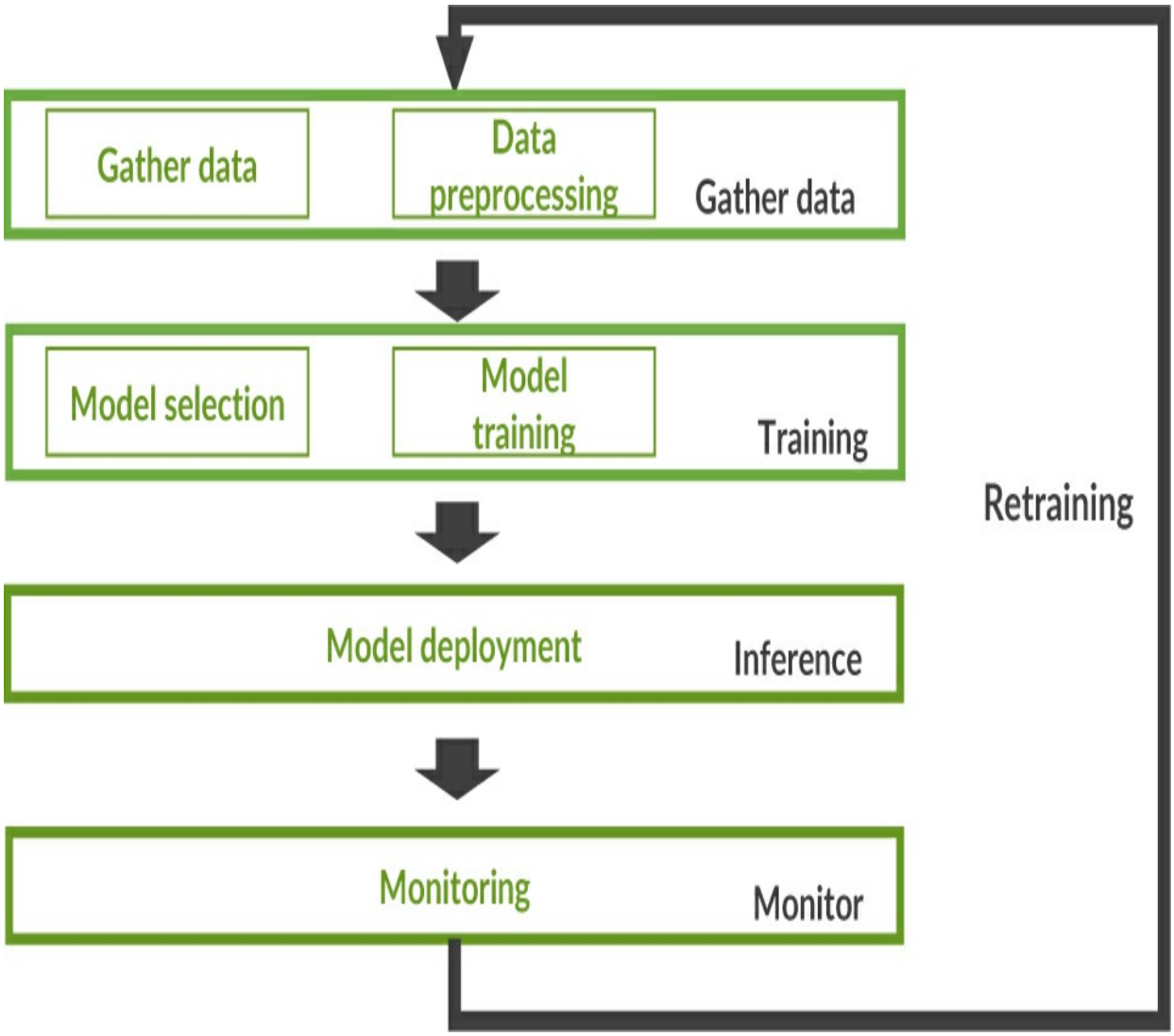


Figure 3-1 ***Stages of processing AI/ML workloads***

The next step is to select models (algorithms) that apply to the prepared data set. A model is trained with the prepared data set to determine its efficiency. The training is run iteratively with different model parameters until the model gives consistent results with high accuracy. Training can be supervised—with the involvement of experts or reinforced.

Once the trained model is determined to be accurate, it is deployed in production systems for external inference requests. It is monitored for inaccuracies and unexpected behavior. Models are trained periodically with newer data sets for fine-tuning and improved accuracy.

Figure 3-2 illustrates different network fabrics connecting to a server. Storage

fabric can either be distributed or dedicated to data centers. Inference fabric acts as frontend and is used for user interactions. Training fabric acts as a backend and is used for training or learning purposes.

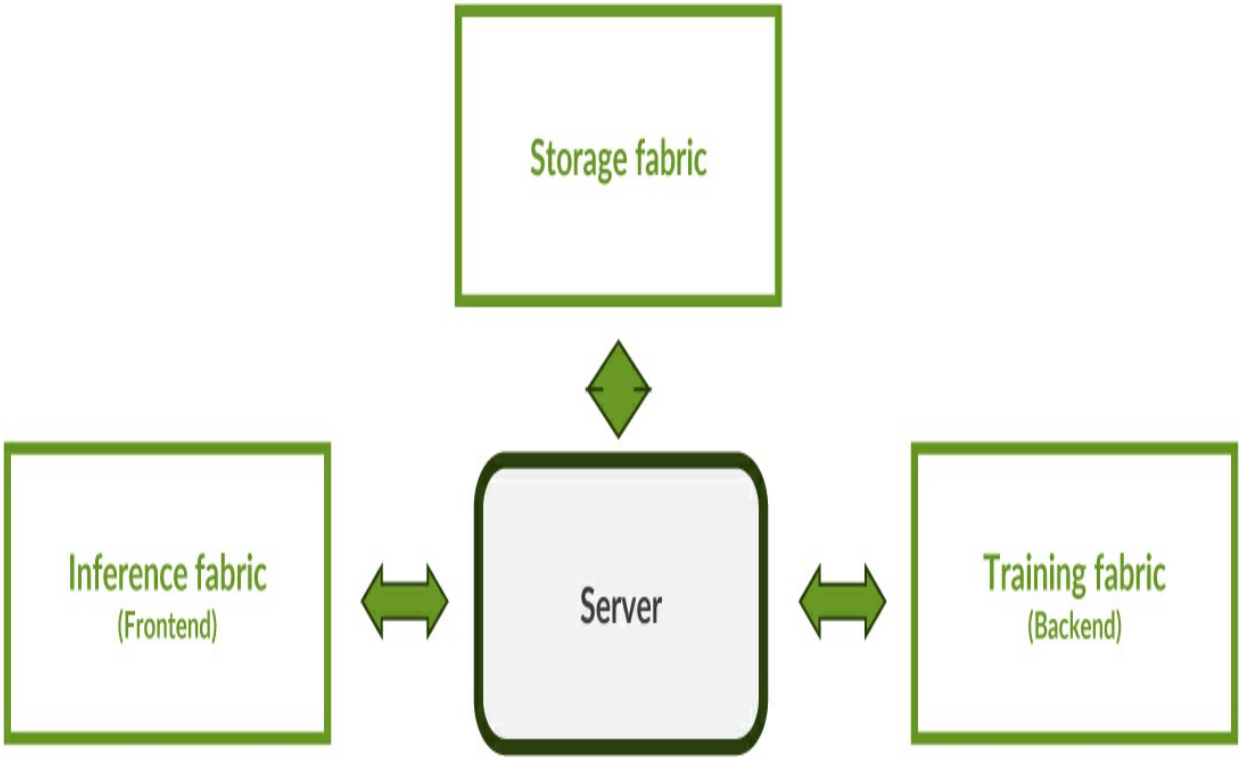


Figure 3-2 Network associations with a server

The following data centers are connected to the server in Figure 3-2:

- **Storage data center:** This data center stores all the gathered data. While smaller data volumes may be stored in the training cluster, large data volumes typically use storage data centers. These data centers have existed for a long time and have continued to evolve. The primary consideration for a storage data center fabric is a low-latency, lossless network. Currently, InfiniBand and Ethernet are the two options for storage. The InfiniBand technology offers superior performance while supporting lossless, low-latency networks. Ethernet is implemented with a three-stage or five-stage Clos network with RDMA over Converged Ethernet version 2 (RoCEv2) features to make it a lossless fabric.
- **Training data center:** Training clusters that are built for research have clusters of GPUs to accommodate large data sets. They work on a

comprehensive list of models to derive a focused list of trained models. These models are used by various applications to provide results to the user. In contrast, the GPU clusters in an enterprise data center are much smaller and run focused models and data sets.

Training data centers, also called backend data centers, have similar considerations to storage data centers. We cover training data center fabric design in this chapter.

- **Inference data center:** The inference data center is the production data center, where the trained model is deployed for external inference requests. The architecture of this type of data center is similar to the architecture of an enterprise, cloud, or telco data center. In most cases, it is similar to a collapsed three-stage or five-stage Clos network.

In this chapter, we investigate the various options for training, or backend, cluster architecture in detail.

Training Data Center Architecture

The network design for AI/ML workloads requires a balance between performance, cost-efficiency, reliability, and scalability. An organization needs a durable network that can cope with the challenging nature of AI/ML calculations while staying flexible to future expansion and technological improvements. This can be achieved by applying optimized topologies, weighing cost trade-offs, and using strong failover strategies.

AI/ML workloads have demanding networking requirements that necessitate high-speed connections between server and leaf switches—typically between 200 Gbps and 400 Gbps. The network design therefore must include specialized switches, high-bandwidth cabling, and topology adjustments to ensure optimal performance and throughput.

Servers such as the Nvidia A100 or H100 are equipped with 8 GPUs, and each GPU may be linked to one or two NICs. While multiple NICs per GPU is feasible, cost considerations around optics, NICs, and server port scalability often favor the implementation of a single NIC per GPU. This approach reduces expenses and eliminates port-level redundancy, removing the requirement for multi-homing on the server side.

The GPU-to-leaf connectivity options include multiple GPUs connected to a single leaf or a one-to-one GPU-to-leaf configuration. For example, for a server housing 8 GPUs, it is essential to have 8 leaf connections, forming what is commonly referred to as a rail-optimized topology. This topology choice is preferred because of its efficiency in handling high-throughput AI/ML workloads; it offers enhanced data transfer capabilities and optimized performance.

There are two design options for GPU-to-leaf connectivity:

- One-to-one GPU-to-leaf configuration, referred to in this book as rail-optimized design (ROD). (There is a similar-sounding concept, rail-only design, where there is no communication between the different rails. We discuss this design in subsequent sections on the network fabric.)
- Multiple GPUs connected to a single leaf, referred to in this book as rail-unified design (RUD).

Figure 3-3 shows the different network ports found on GPU-based servers. GPUs use specific ports on a server to communicate to other GPUs across servers. These servers are mainly used for training in training networks. Certain ports within a server are used for CPU communication across servers or for northbound connections. These ports are mainly used for inference and are part of frontend networks. In addition, some ports are used for Non-Volatile Memory Express (NVMe) communication for fast data transfer.

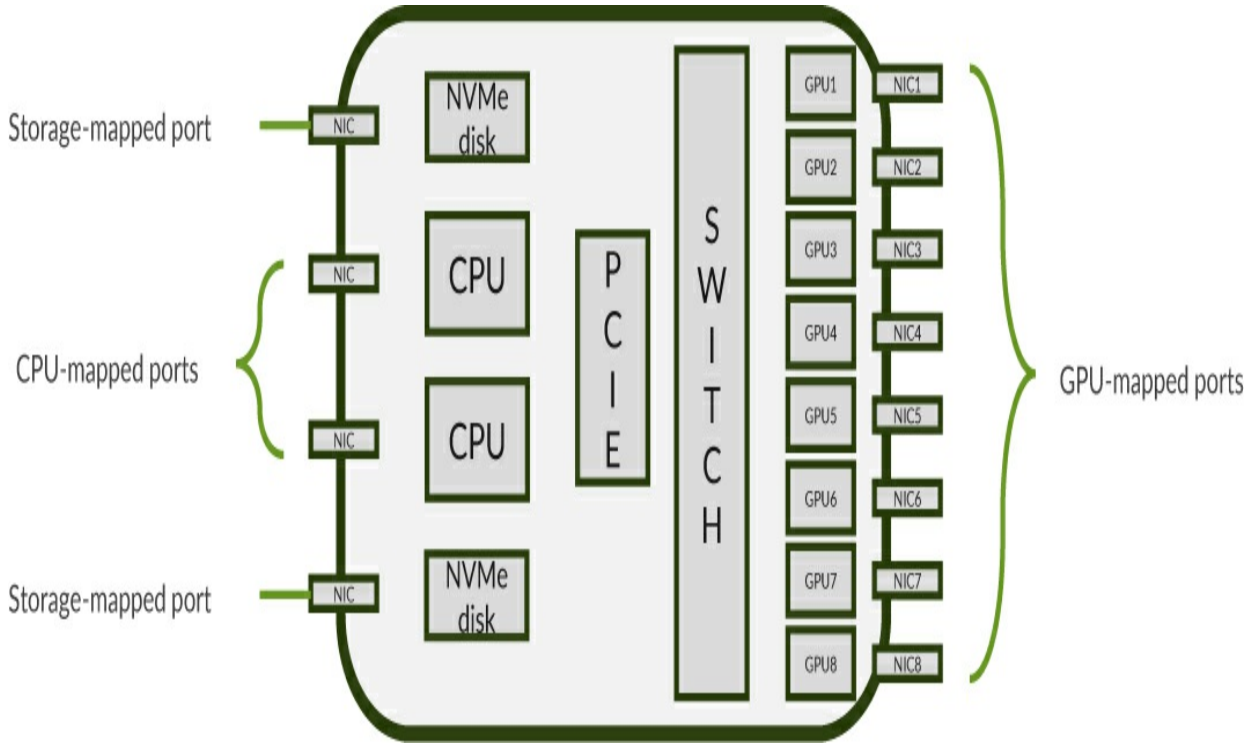


Figure 3-3 *NICs in a GPU-based server*

Rail-Optimized Design (ROD)

Servers used in AI datacenters, for example DGX series, HGX series from Nvidia, SuperMicro GPU Super server series, MI series from AMD, etc; each have an internal switch that connects the GPUs. This internal switch is used for any communication within the server, and it is faster and provides higher performance compared to an external switch. However, when the workload processing requires more than 8 GPUs, the load is shared across servers, resulting in east–west traffic. While the GPUs within a server communicate via the internal switch, the NICs mapped to the GPUs can be connected to multiple leaf switches for low latency, with one hop communication between servers.

Each GPU in a server is considered to be part of a different rail. Across servers, a GPU number is mapped to a rail of the same number. Figure 3-4 displays the mapping of GPUs to a rail.

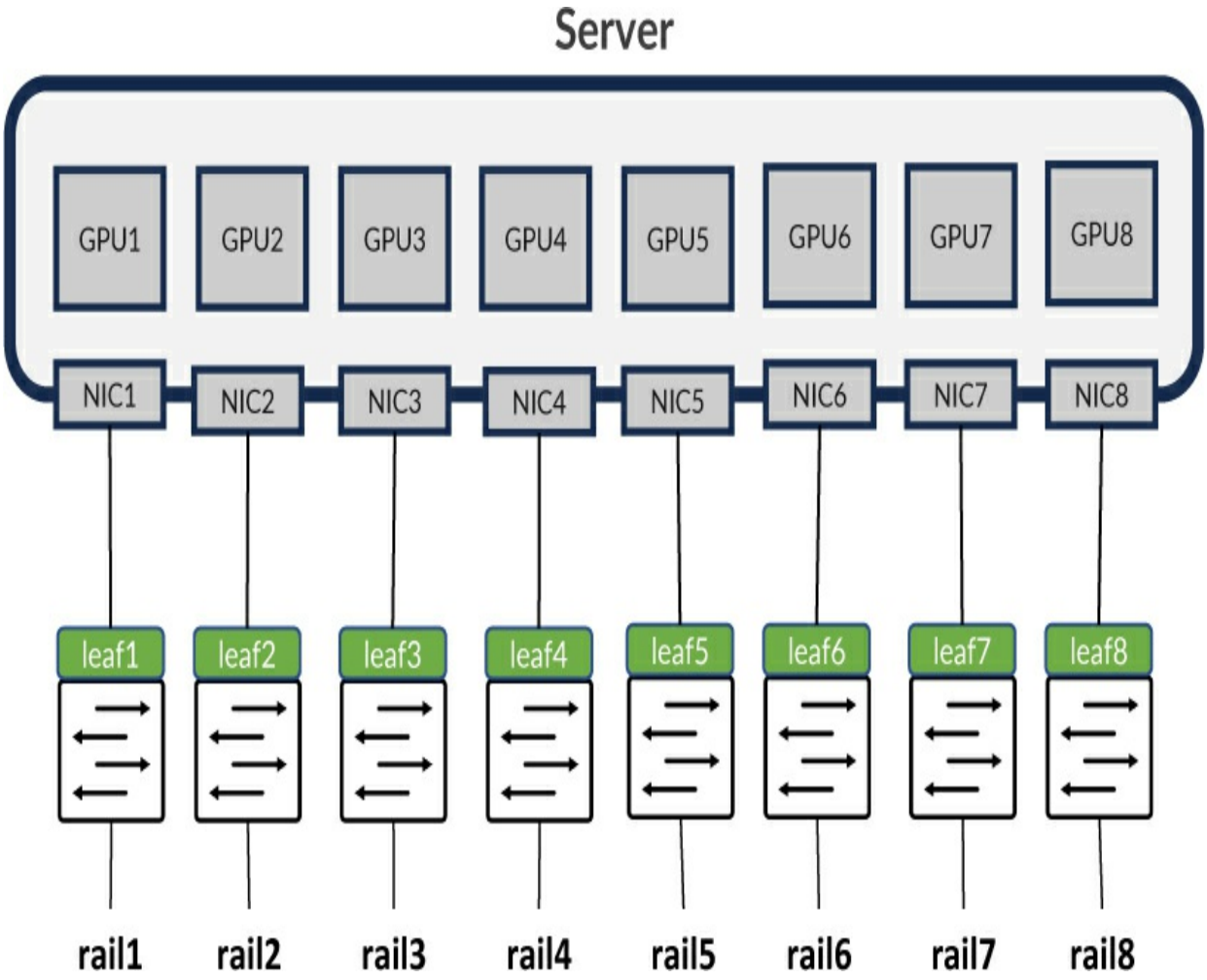


Figure 3-4 *Rail-optimized topology*

To keep the latency within a rail to a minimum, each GPU is connected to a leaf switch; this is referred to as rail-optimized topology. Eight leaf nodes connecting to 8 different GPUs is referred to as a row, or a stripe. Each leaf switch can establish connections with several servers. For instance, a 64 × 400 Gbps switch with a 1:1 oversubscription ratio can have 32 × 400 Gbps links toward the servers and an additional 32 × 400 Gbps links toward the spine, as illustrated in Figure 3-5.

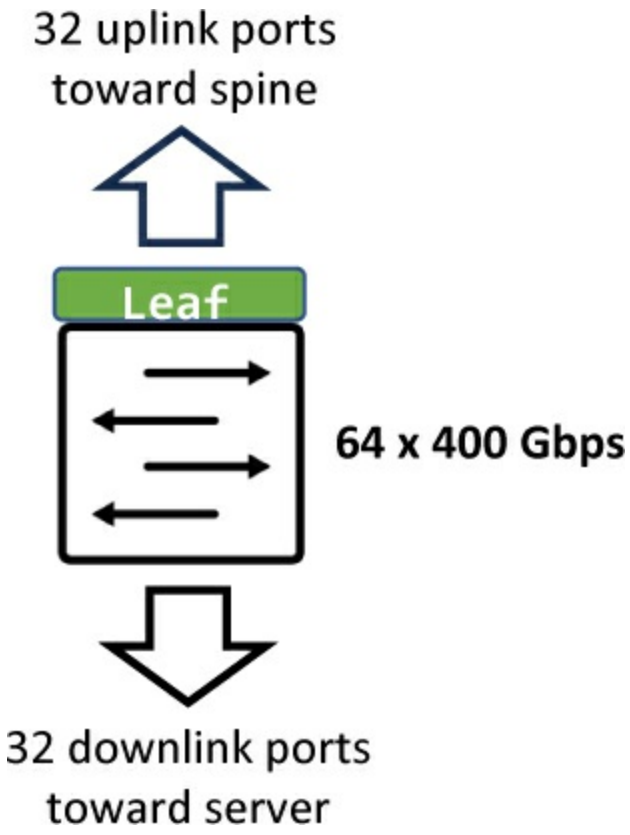


Figure 3-5 *Leaf switch with 32 uplinks and 32 downlinks*

A cluster of 256 GPUs (32 servers × 8 GPUs per server) can be built using 8 leaf switches with 32 × 400 Gbps downlinks per switch. Alternatively, using 8 leaf switches with 64 × 400 Gbps downlinks per switch can enable the creation of a larger cluster, accommodating up to 512 GPUs (64 servers × 8 GPUs per server). For a cluster of this size, the infrastructure can operate efficiently without the spine layer, as shown in Figure 3-6.

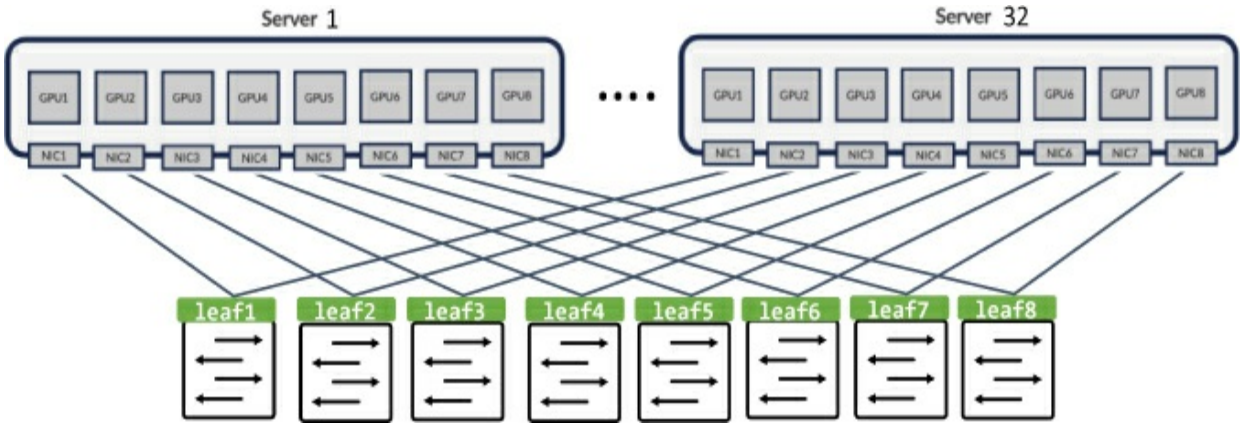


Figure 3-6 *Rail connections with multiple servers*

Note that this topology enables switching only within the same rail—in other words, on the same GPU number on all the servers through the leaf; this is called intra-rail communication. There is minimal latency within a rail. For communication between different GPUs across rails, the traffic needs to flow via the spine layer, as shown in Figure 3-7. This is called inter-rail communication, and its latency is doubled compared to the latency in intra-rail communication.

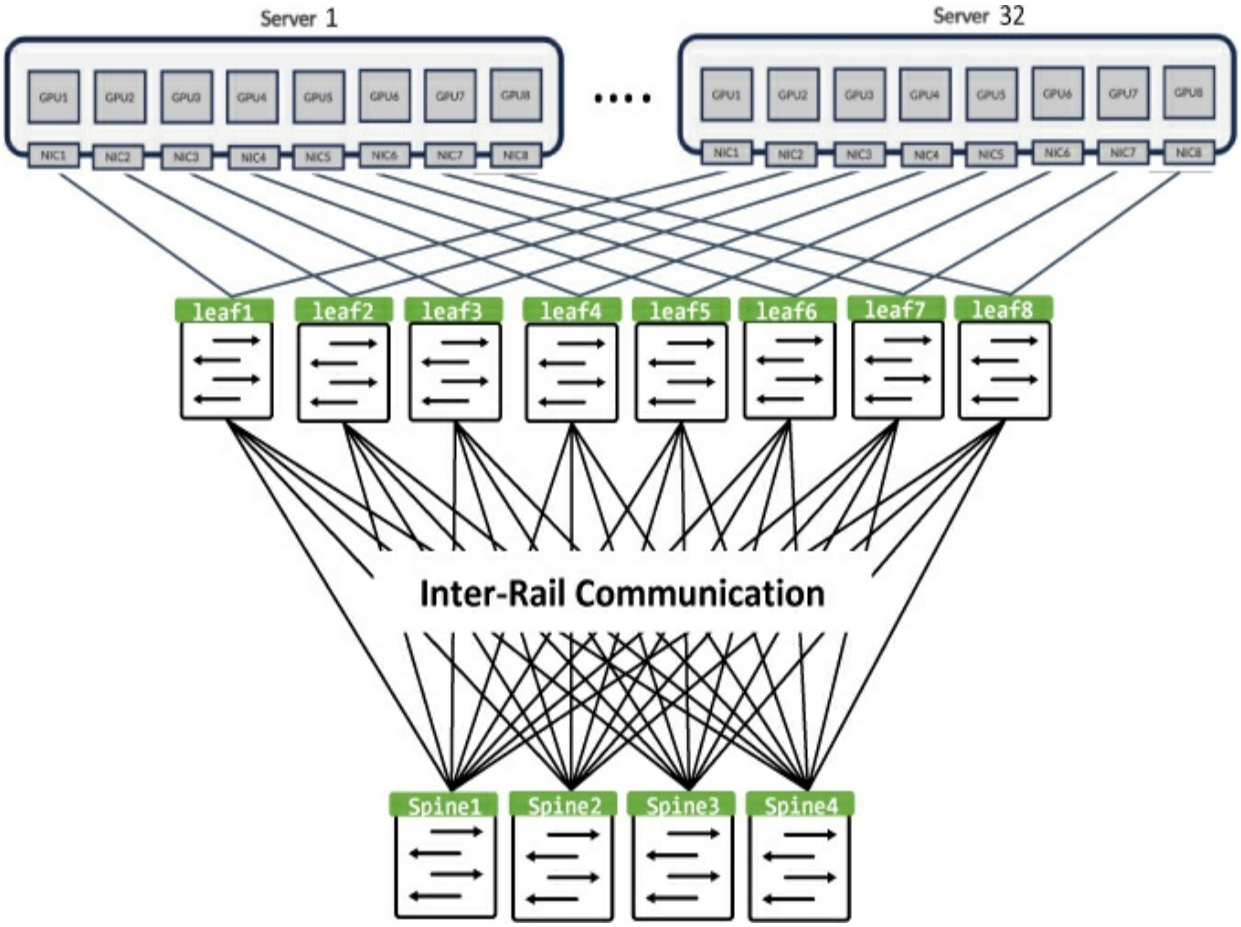


Figure 3-7 *Inter-rail communication*

With 8 leaf switches and 4 spine switches, each 64 × 400 Gbps, the cluster can be scaled to 32 servers (or 256 GPUs). To scale additional servers or GPUs, further rows of leaf and spine switches must be added. With two such rows of leaf switches—that is, 16 leaf switches—and 8 spine switches, the cluster can scale to 64 servers (or 512 GPUs), as shown in Figure 3-8.

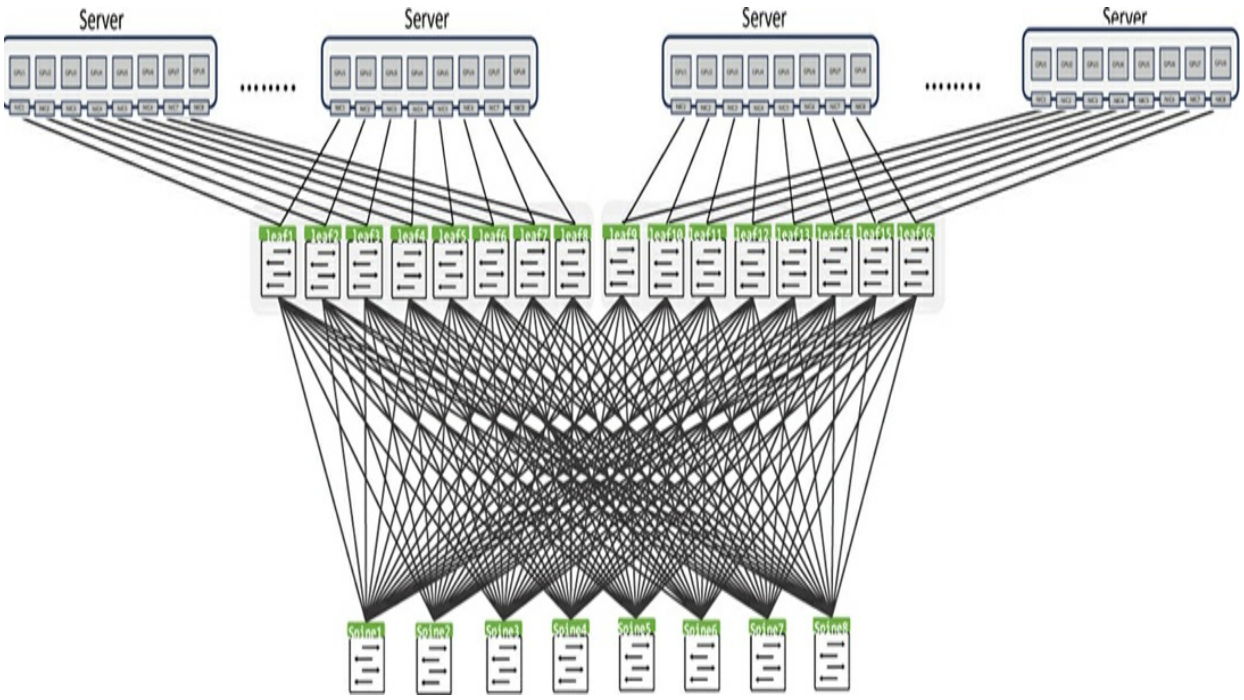


Figure 3-8 *Connecting two rows of rails*

To scale further, either the number of spines needs to increase or a chassis-based spine must be used. However, increasing the number of spines will create load-balancing and management challenges. Figure 3-9 illustrates the GPU scale supported with a three-stage Clos fabric. It uses 64 × 400 Gbps switches at each layer.

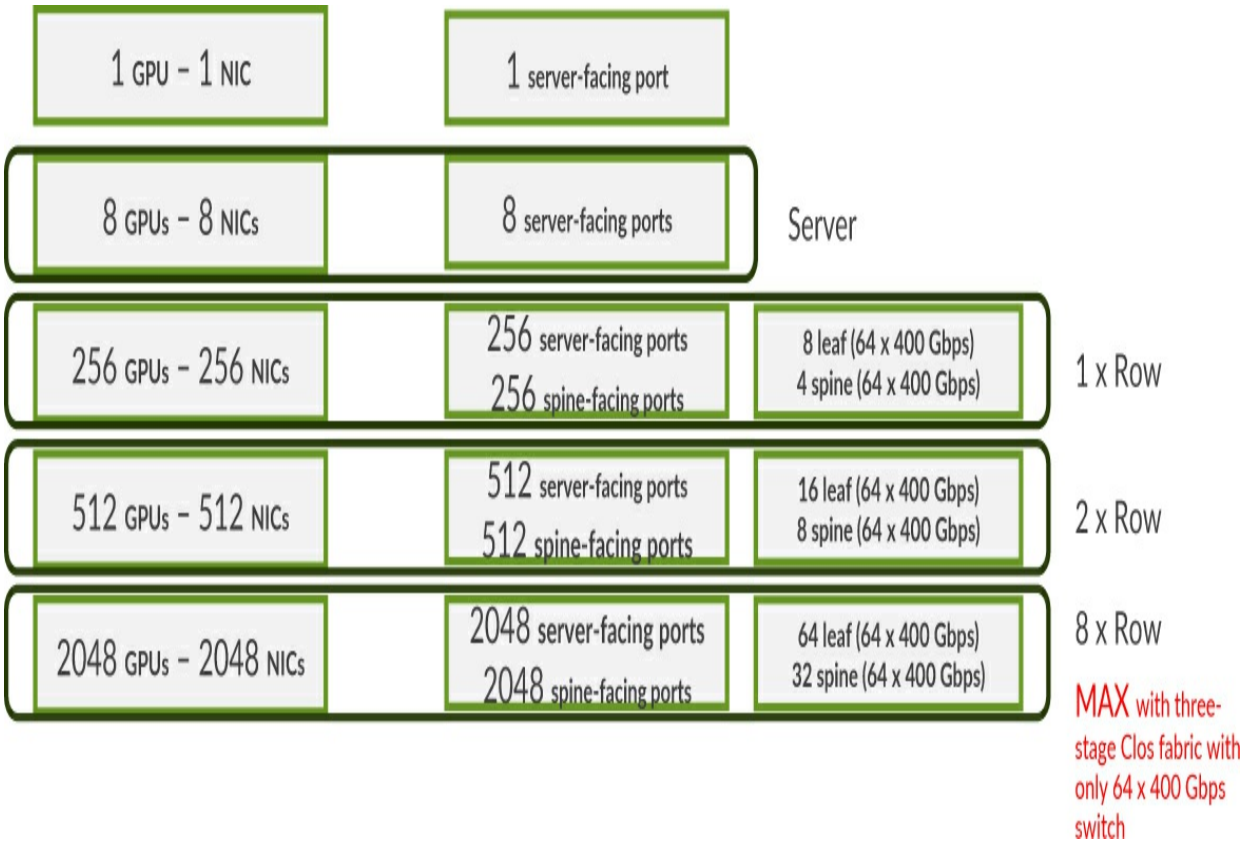


Figure 3-9 *GPU scale supportability with three-stage Clos fabric*

Now, if the spine is a chassis-based system with, let's say, 256 × 400 Gbps, the number of spines needed will be reduced to one-fourth compared to 64 × 400 Gbps spines, as illustrated in Figure 3-10.

Figure 3-10 *GPU scale supportability with a chassis-based spine*

Often, large AI/ML clusters can scale up to 32K, 64K, or 128K GPUs. To achieve such a cluster size, you may need to deploy 8K, 16K, or 32K leaf switches. This requires a multistage Clos architecture such as a five-stage or seven-stage Clos fabric.

A combination of spine-connected rows is called a block, or brick. The blocks, or bricks, are interconnected via a super spine. In addition, the communication between blocks may not be 1:1 oversubscribed, if correctly

controlled via server applications.

There are several ways to scale up to 32K, 64K, or 128K GPUs:

- Use five-stage or seven-stage Clos architecture.
- Use chassis systems with high port density at the spine and super spine layers.
- Apply oversubscription at the super spine layer.

Figure 3-11 illustrates the block, or brick, for a five-stage Clos fabric with enough ports for super spine connectivity.

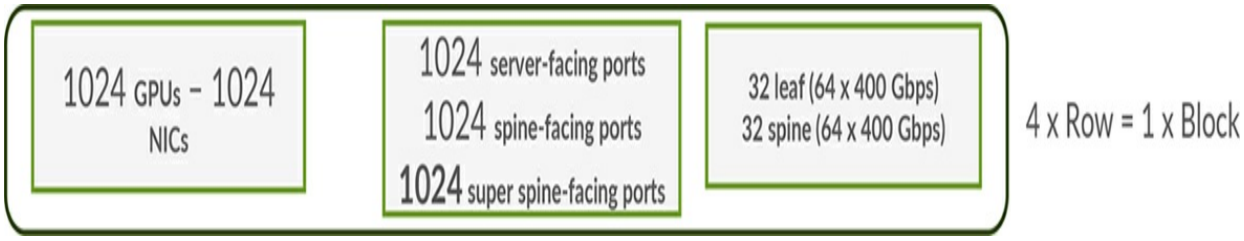


Figure 3-11 *GPU scale supportability per block with five-stage Clos fabric*

With 1K GPUs per block, we need 32 blocks to make a cluster of 32K GPUs. Figure 3-12 illustrates how cluster sizes are incremented with different oversubscription ratios at the super spine layer. Note that the number of super spines is consistent.

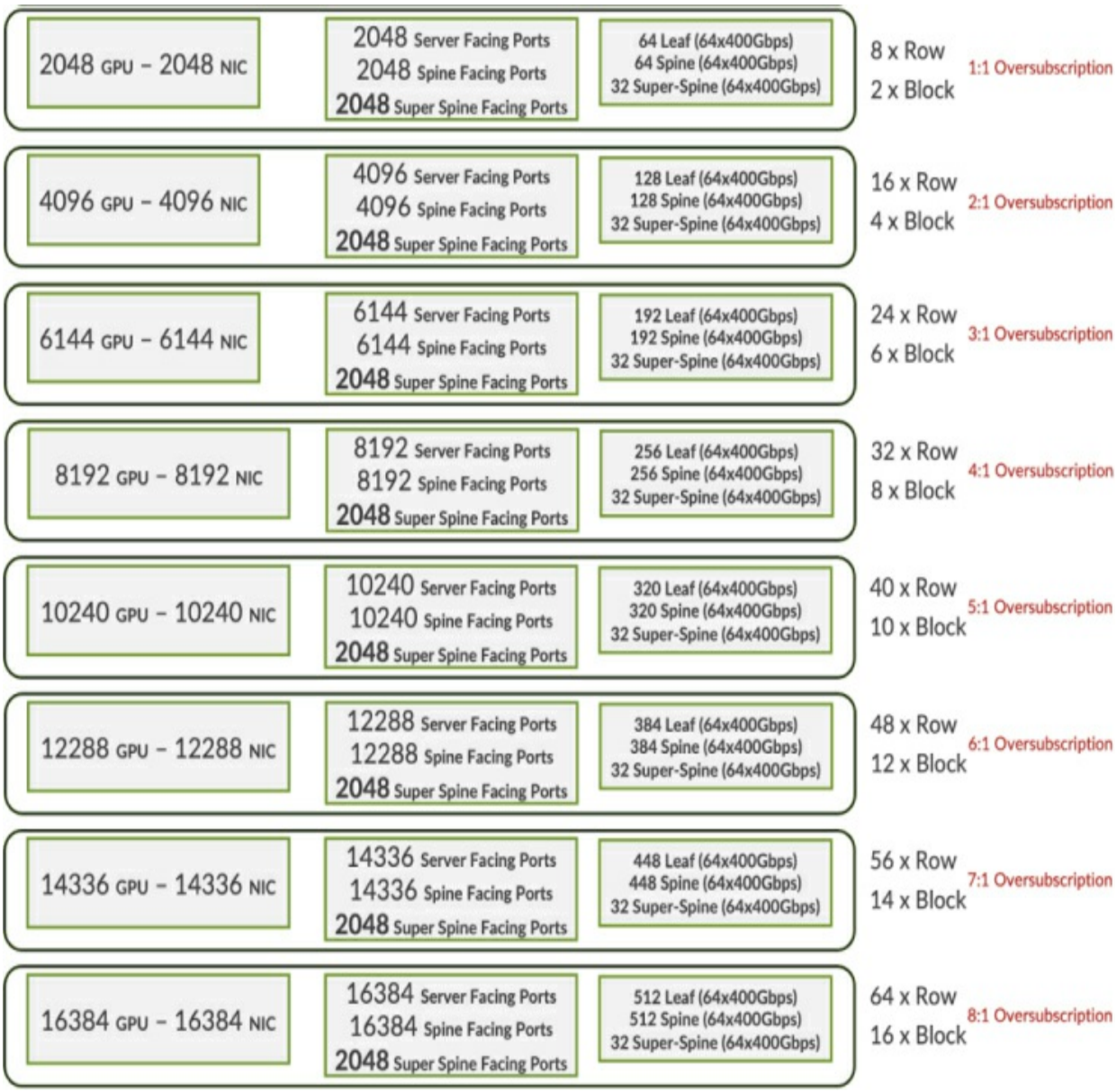


Figure 3-12 *GPU scale supportability with different oversubscription ratios*

Currently, some companies are evaluating use of an oversubscription ratio up to 7:1 at the super spine layer. The challenge with higher oversubscription ratios is that the likelihood of congestion at the super spine layer also increases. This congestion requires implementation of Data Center Quantized Congestion Notification (DCQCN)—a congestion control mechanism for RoCEv2, as discussed in Chapter 7, "RoCEv2 Transport and Congestion Management."

It is also possible to create large clusters by using a chassis-based system for

the super spine layer. With super spine switches of higher port capacity, it is possible to create a 16K cluster without increasing the number of super spines, as shown in Figure 3-13.

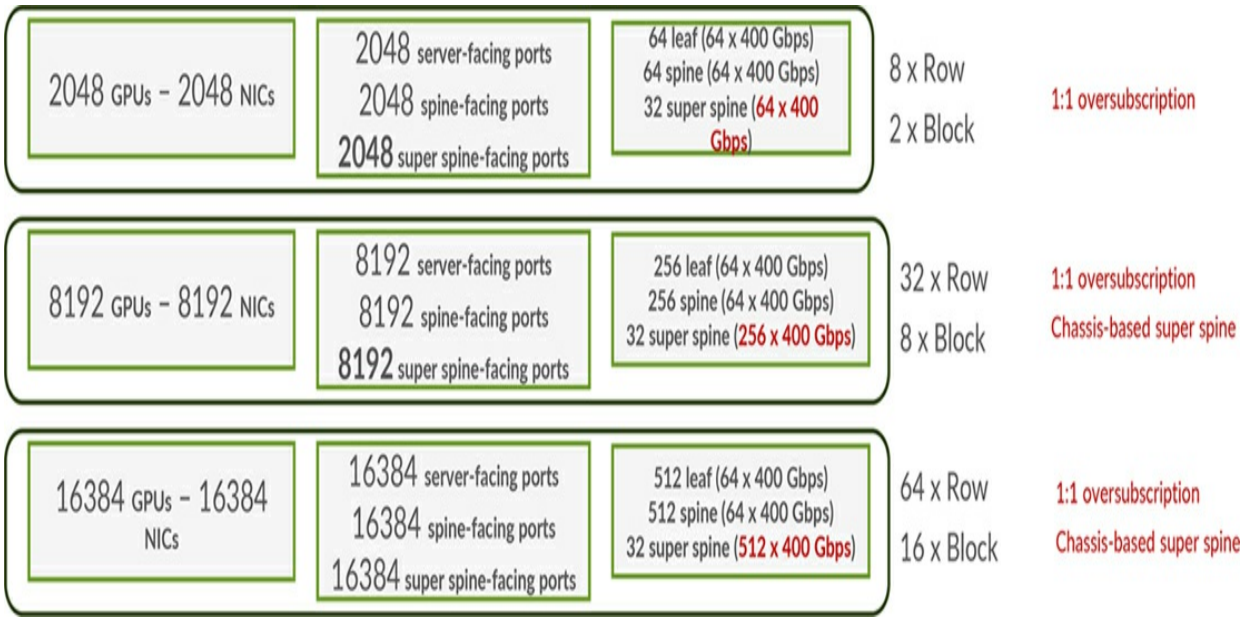


Figure 3-13 *GPU scale supportability with five-stage Clos fabric and a chassis-based super spine*

We already discussed inter-rail traffic. Companies are trying to evaluate the requirement of inter-rail traffic in the training clusters. With inter-rail traffic, each of the GPU servers has 8 GPUs and an internal high-bandwidth switch for communication between the GPUs within the server. The idea is that if inter-rail traffic can be handled via the switch internal to the server, we do not need inter-rail connections in the fabric. This design, referred to as rail-only design, reduces the budget of an AI/ML data center, as illustrated in Figure 3-14.

Rail-only design can also be used in extending the rails across the stripe. In this case, the spine layer will be used to connect different stripes. The spine-to-leaf connectivity needs to be 32 × 400 Gbps for 1:1 oversubscription ratios. The scaling constraints remain the same as those discussed previously, but the spine also follows the rail connectivity. This kind of topology can make the topology simpler in cases where there is no communication between the rails. Troubleshooting is also often simpler because the faults in one rail do not impact the other rails.

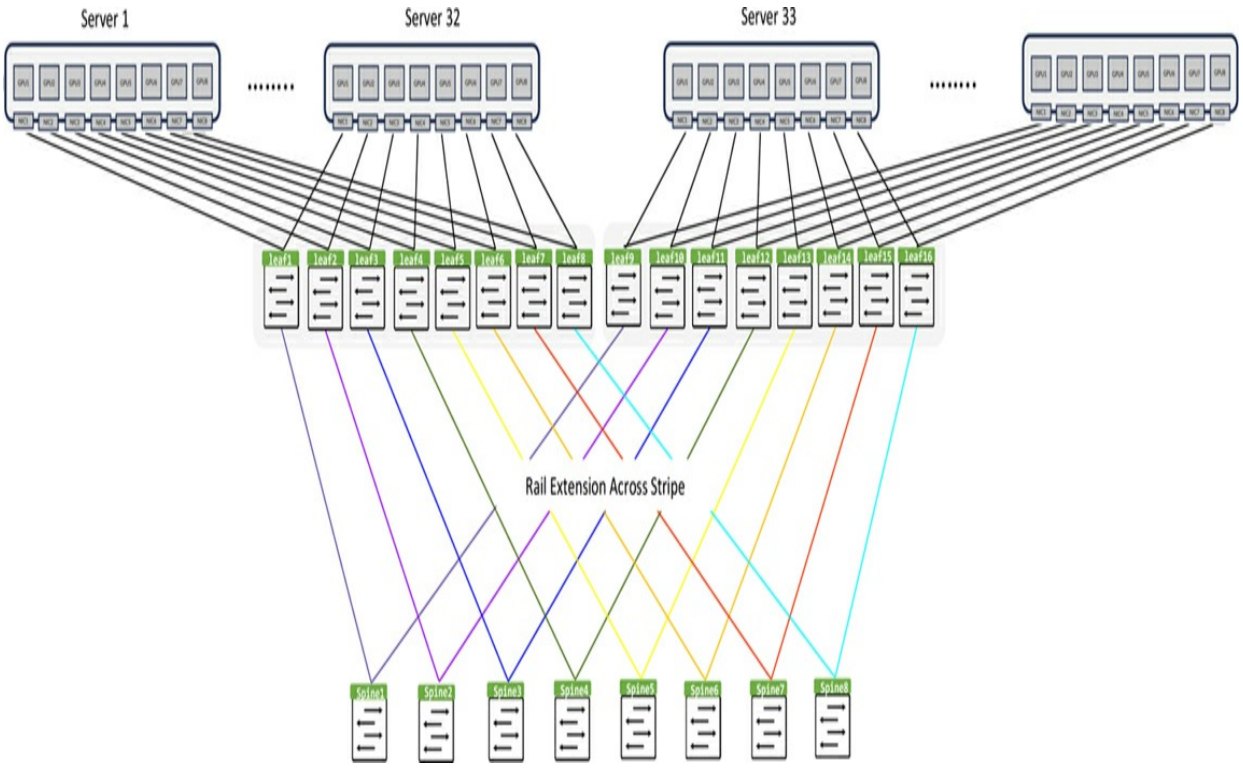


Figure 3-14 *Rail-only design*

The benefits and adoption of rail-only topology are still to be determined at the time of writing of this book.

Rail-Unified Design (RUD)

Another design option is to connect multiple GPUs in a server to a single leaf. You can connect all 8 GPUs of a server to one leaf switch, you can connect 4 GPUs to two leaf switches, and so on, as illustrated in Figure 3-15. The advantage of this topology, called rail-unified design (RUD), is simplified cabling, easier scaling, and well-established architecture. The challenge is that if the single leaf goes down, the server is completely isolated.

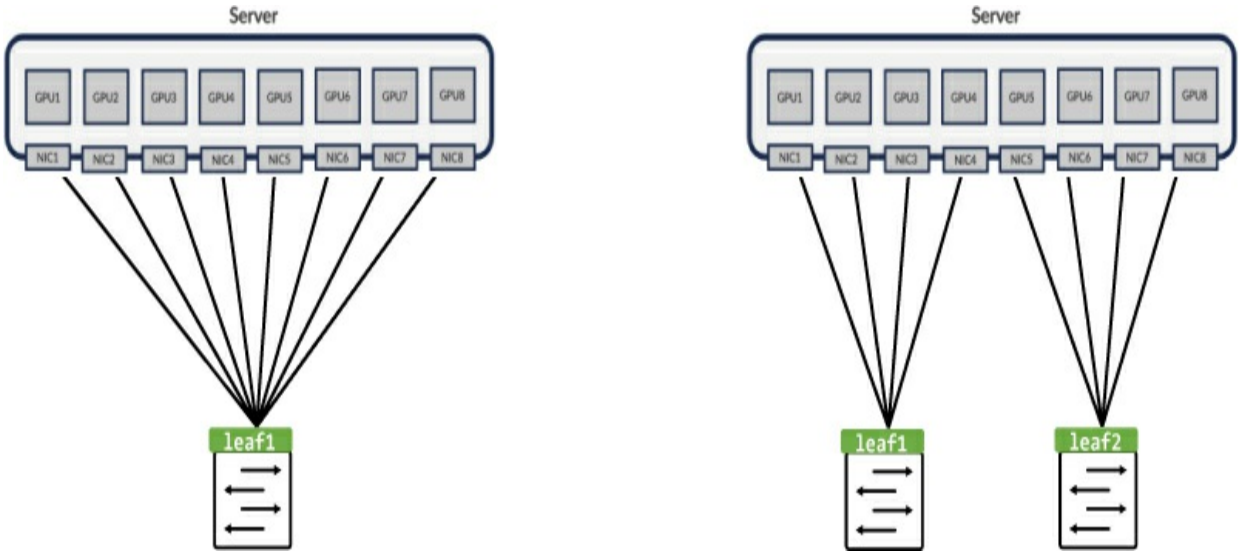


Figure 3-15 *Rail-unified design*

As discussed earlier, the GPUs within a server communicate via the internal switch. Therefore, local GPU traffic does not use the links from the leaf toward the server. Four servers connected to the same leaf will have the least latency. Intra-rail and inter-rail communication with up to 4 servers has one-hop latency. Traffic from the GPUs on servers other than these four servers passes through the spine, with a latency of two hops, as shown in Figure 3-16.

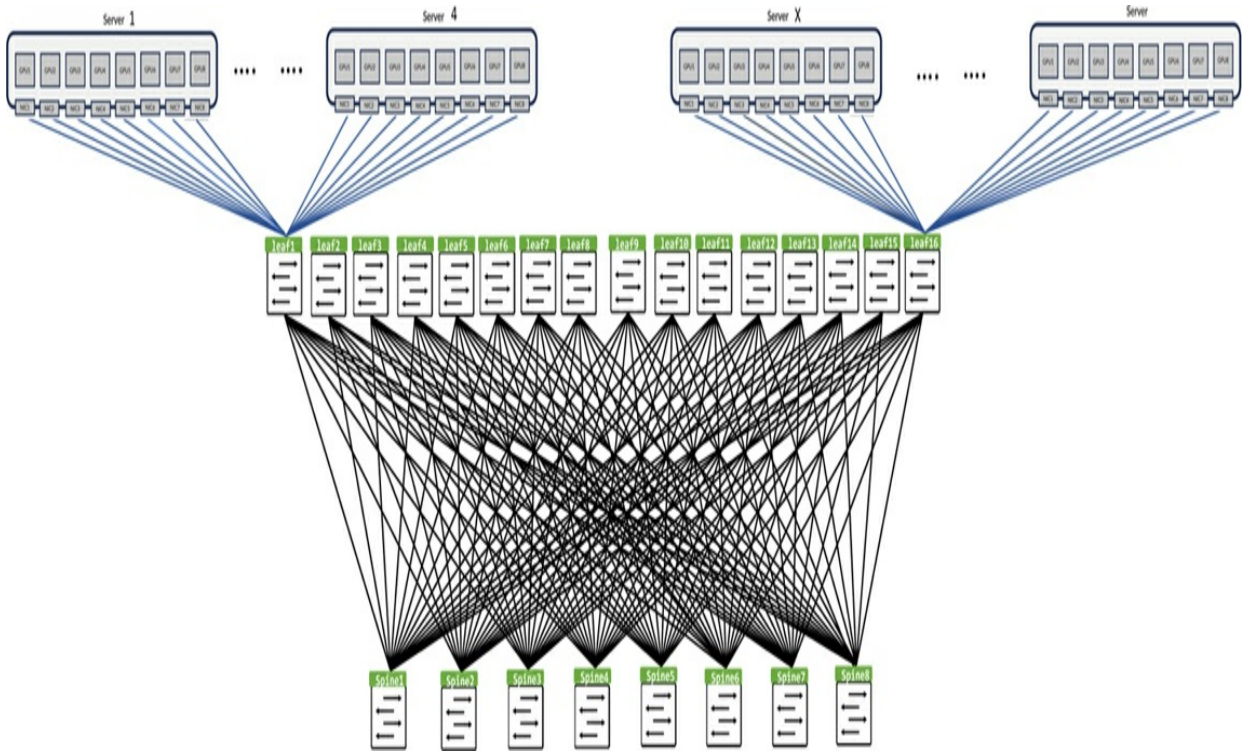


Figure 3-16 *Rail-unified design with multiple servers*

Because each leaf has multiple rails in this topology, it must be able to segregate rail traffic. This requires additional features such as deterministic path forwarding, as shown in the Figure 3-17, where the green-colored highlight represents a rail. (If you are reading a printed copy of this book, you will see a shade of gray here, but you can still see the highlighted rails.)

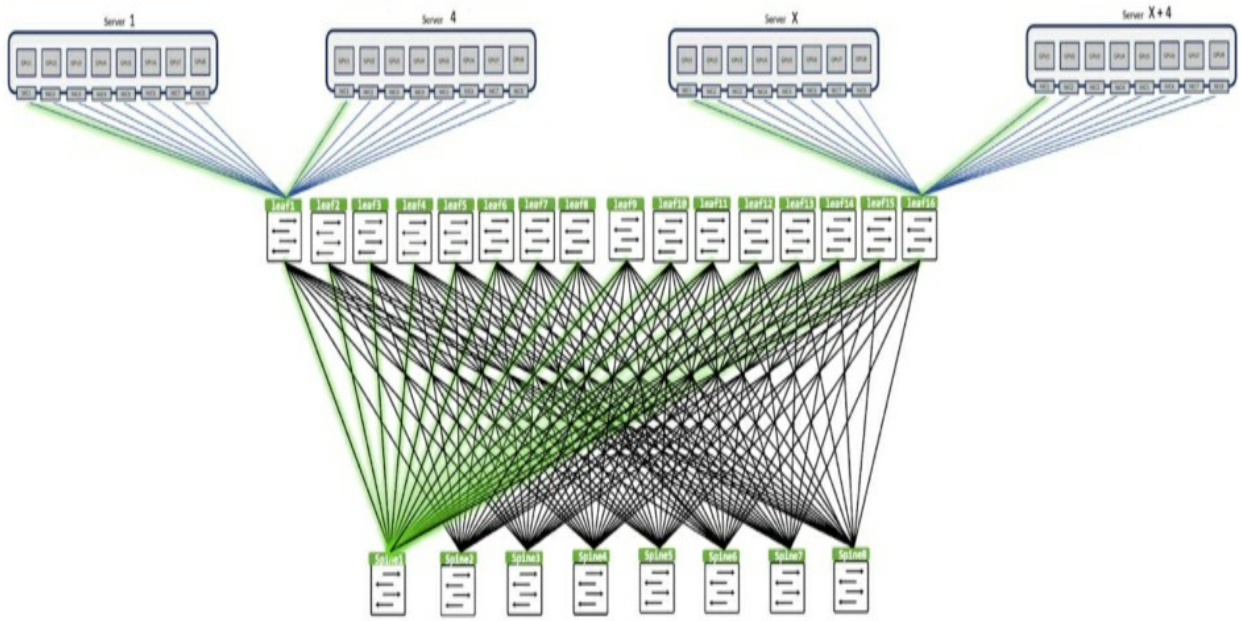


Figure 3-17 *Rail-unified design with deterministic path forwarding*

The rail-unified design topology can be quite attractive to customers that do not plan to use the internal switch for GPU communication within a server and instead use a higher-throughput switch. For example, the Nvidia NVSwitch offers fast throughput, at 900 Gbps, for GPU-to-GPU connectivity. However, this topology comes at a cost compared to an external datacenter switch. Figure 3-18 illustrates using a leaf switch for GPU communication within the same server.

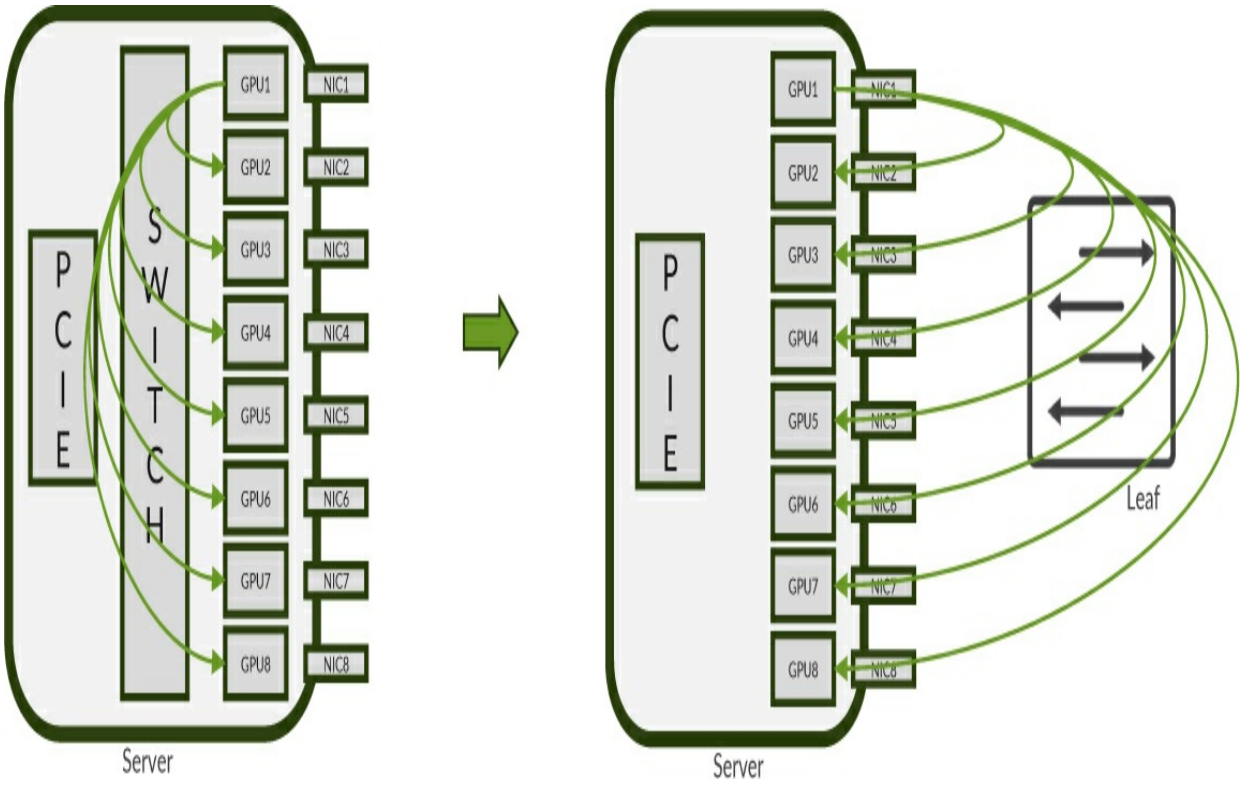


Figure 3-18 *Intra-GPU communication options*

Rack Design

When designing a data center rack, various factors must be considered, including the following:

- Rack capacity
- Power budget of the rack
- Cooling requirements
- Cabling requirements

A typical rack in a data center is 19 inches wide, 36 inches deep, and 42U tall. The Nvidia DGX H100 server is 19 inches wide, 35.3 inches deep, and 8U tall, whereas the DGX A100 is 6U tall. A single rack can host four or five DGX H100 servers and a leaf switch, requiring a power budget between 48 kW and 60 kW. This is very high compared to the average power budget of 20 kW to 25 kW in current data centers. We cover power and thermal management in Chapter 5, “Thermal and Power Efficiency Considerations.”

There are three data center designs for the position of leaf switches:

- Top-of-rack
- Middle-of-row
- End-of-row

In addition, for rail-optimized design, one server needs to connect to 8 different leaf switches.

Top-of-Rack

In the top-of-rack (ToR) design, the switch is installed at the top in each rack. Figure 3-19 illustrates a ToR design that has a row of 8 racks, where rack has 4 servers and 1 leaf switch. This makes a cluster of 32 servers and 256 GPUs.

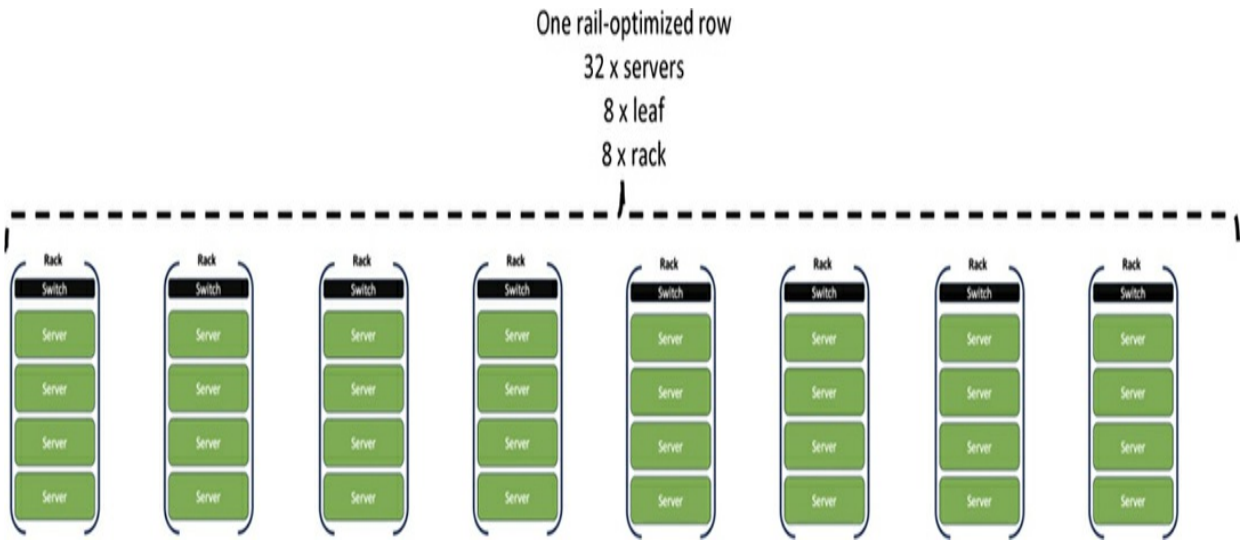


Figure 3-19 *Top-of-rack design*

With this design, the spine can be installed either in a separate rack or in the same rack. A spine in the same rack can be used with the block, or brick, design. Figure 3-20 illustrates a ToR design with the spines in separate racks. Keep in mind that the number of spines and racks depends on the design requirements.

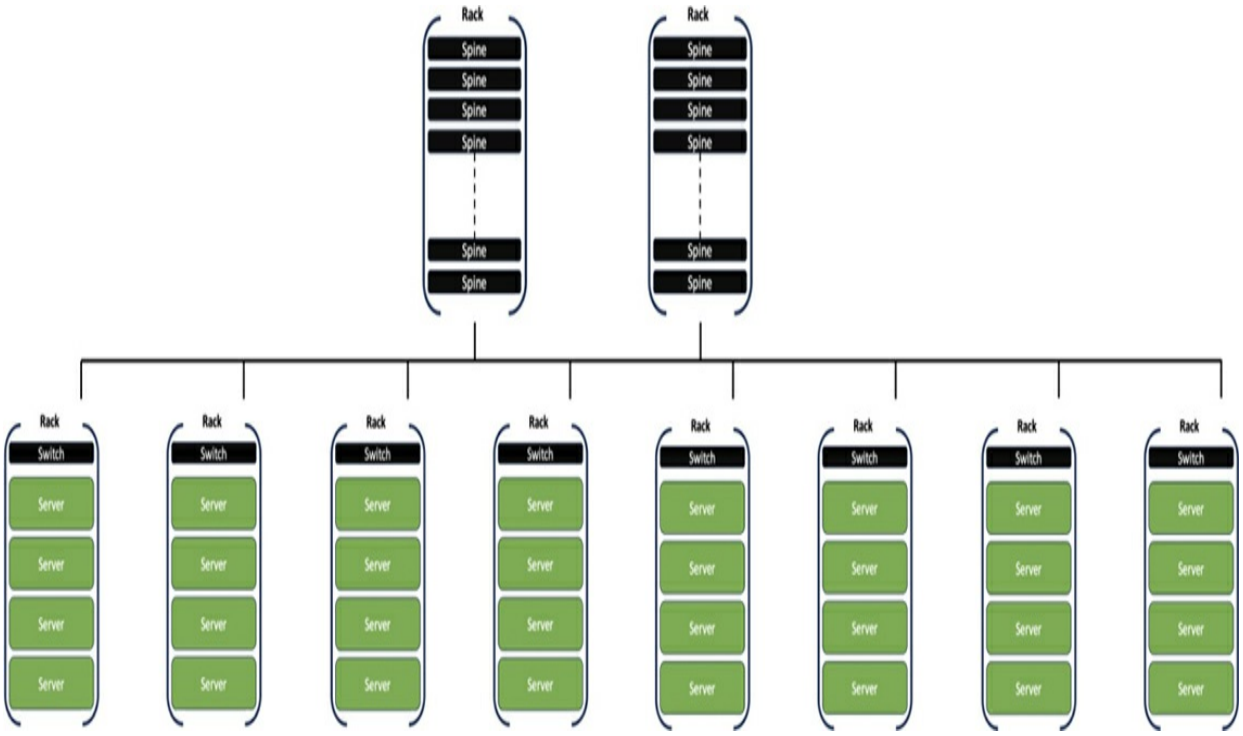


Figure 3-20 *Top-of-rack design with the spines in separate racks*

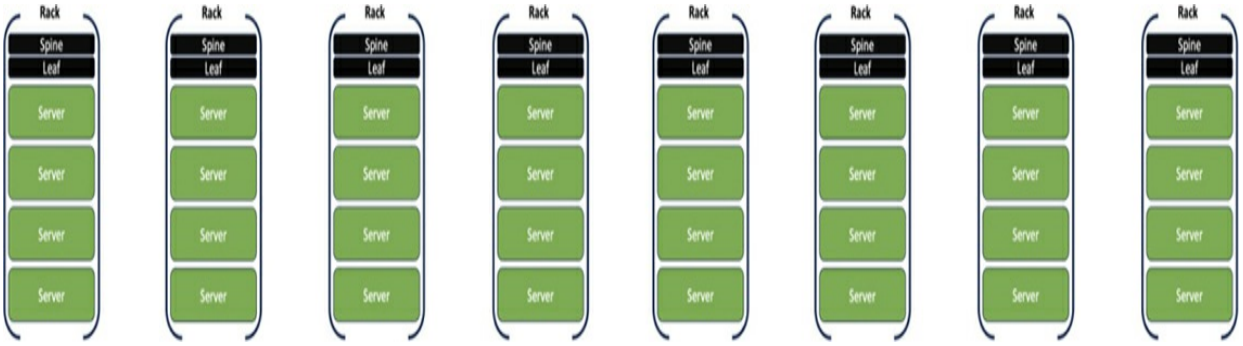


Figure 3-21 *Top-of-rack design with the spine in the same rack*

The topology illustrated in Figure 3-21 has an equal number of spine and leaf switches. However, it is possible for only a few racks to have spines based on cluster size and oversubscription requirements. This topology requires cable lengths to run within the rack as well as across the 8 racks. The power budget and cooling requirement of the rack increase with switches on the same rack. This type of topology is advantageous for an RUD design, though. The cabling is easier, and copper-based Direct Attach Copper (DAC) or Active Electrical Cable (AEC) cables can be used for cost benefit.

Middle-of-Row

In the middle-of-row design, a switch is installed in the middle rack, between the server racks, as shown in Figure 3-22. The design in this figure has a row of 8 server racks, each rack with 4 servers. It makes a cluster of 32 servers and 256 GPUs. Each server connects to 8 leaf switches, which are installed in their own racks in the middle of the row. The leaf switches can be in one rack or more than one rack for better failure handling. Spine switches can be installed on the same rack as the leaf switches or on a separate rack.

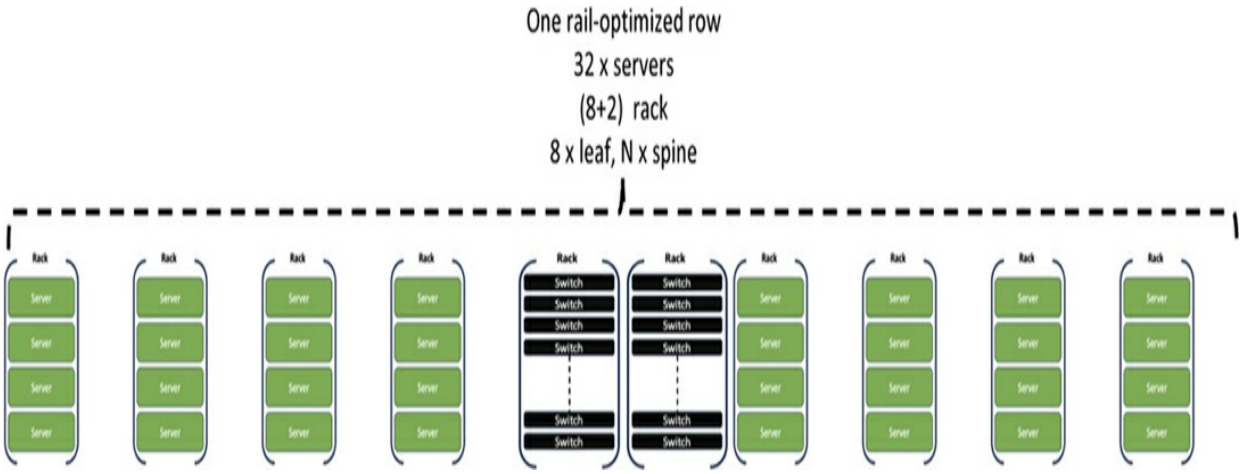


Figure 3-22 *Middle-of-row design*

This topology requires the cable lengths to run from servers to leaf switches across a maximum of five or six racks. The power requirement of a switch does not add to the power budget of the rack. However, this topology requires more rack space, especially when spines are installed in their own racks, separate from the leaf switches.

End-of-Row

In the end-of-row design, the leaf switches are installed in a rack at the end of the row, as shown in Figure 3-23. This figure shows a row of 8 server racks, each rack with 4 servers. It makes a cluster of 32 servers and 256 GPUs. Each server connects to 8 leaf switches, which are installed in separate racks at the end of the row. The leaf switches can be in one rack or more than one rack for better failure handling. Spine switches can be installed on the same rack

as the leaf switches or on a separate rack.

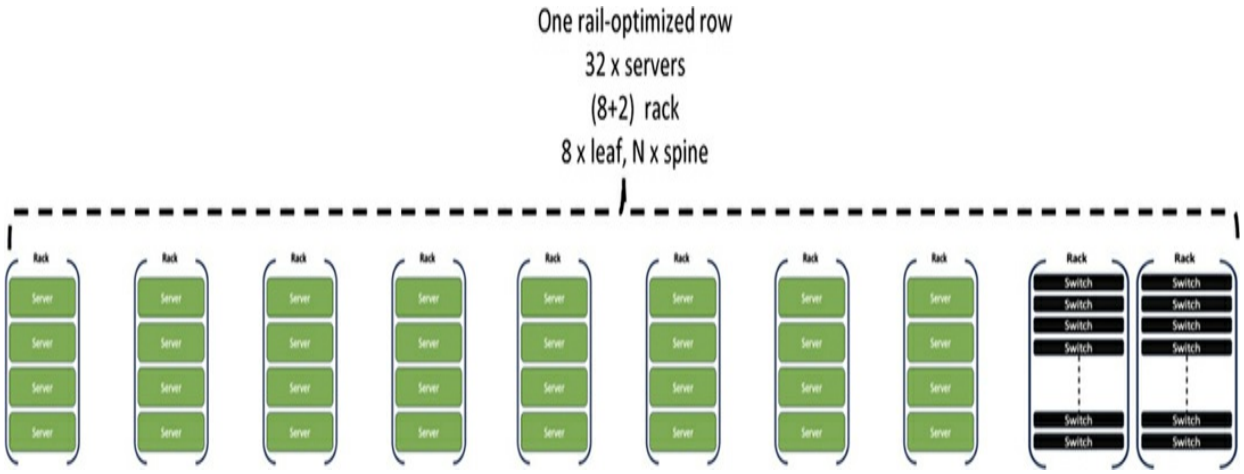


Figure 3-23 *End-of-row design*

This topology requires cable lengths to run from server to leaf across a maximum of 10 racks. The power requirement of a switch does not add to the power budget of the server rack. However, it requires more rack space, especially when spines are installed in their own racks, separate from the leaf switches.

Design Comparison

Table 3-1 provides a comparative summary of the rack design options for rail-optimized design.

Table 3-1 *Leaf Placement Comparison*

	Top-of-Rack	Middle-of-Row	End-of-Row
Rack space requirement	Less	More	More
Power requirement per rack	More	Less	Less
Cooling requirement per rack	More	Less	Less
Cabling requirements within the rack	Varying size connecting servers to leaf switches in each rack	No cabling within a rack	No cabling within a rack
Cabling requirements across the racks	Each server connects to a leaf switch in each rack that is part of a row.	Varying sizes are used from different racks, based on the distance from the network rack. Each server in the rack needs cable of the same size to reach the leaf switch.	Varying sizes are used from different racks, based on the distance from the network rack. Each server in the rack needs cable of the same size to reach the leaf switch.
Maximum cable length	To span 8 racks	To span 6 racks	To span 10 racks

When deploying the rail-unified design, the top-of-rack design is ideal because of the low rack requirements and low cabling requirement. Further, copper-based DAC or AEC cables can be used for cost benefits.

Scheduled Fabric

Scheduled fabric is yet another architecture that Cisco is developing with Silicon One and that Broadcom is developing with Jericho3-AI and Ramon3. This architecture expands the chassis cell-based packet handling to the fabric.

As illustrated in Figure 3-24, each leaf switch acts as a line card, and each spine switch acts as a backplane. The packets arriving at the leaf are split into smaller cells of 64, 128, or 256 bytes, much like a switch or a router. The cells assemble to the spine that acts as the backplane. Finally, the cells are assembled to the egress leaf, and the packet is forwarded to the destination server.

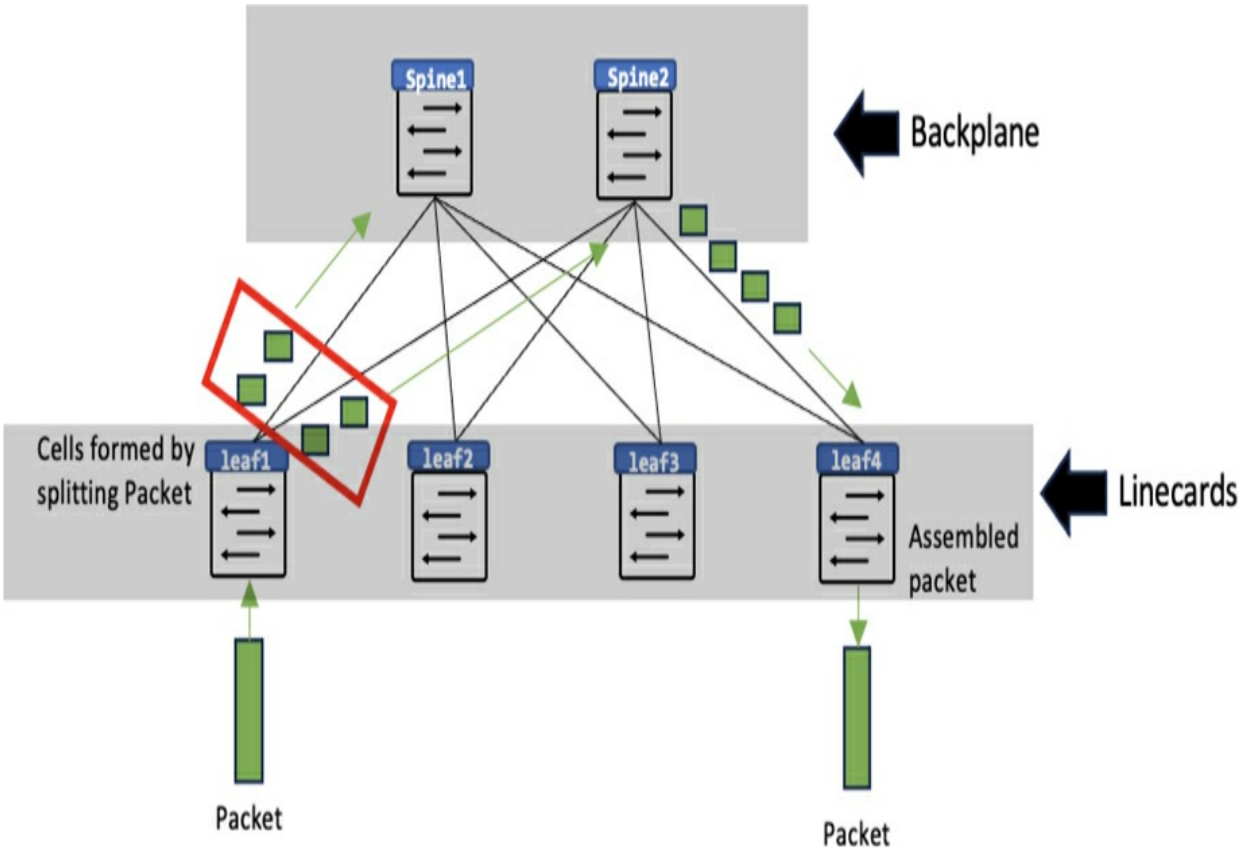


Figure 3-24 *Scheduled fabric*

This architecture as a number of advantages, including the following:

- **Better congestion handling:** This architecture implements a virtual output queue across the fabric.
- **Better utilization of fabric links:** This architecture sprays cells across multiple links.
- **Improvements with latency:** As the cells are transmitted over the fabric, cells split and assembly is happening only once in the fabric instead of each node doing it separately.

However, this is a new architecture and must be evaluated against the following constraints:

- Scaling constraints
- Link or node failure handling
- Latency with congestion
- Vendor lock-in for a block or brick (or pod).

Virtual output queueing (VOQ) is a technique used to address head-of-line blocking in switching devices. With VOQ, the buffer at each input port maintains a separate virtual queue for each egress port. Consequently, congestion on an egress port does not affect traffic toward other egress ports. This is achieved using a scheduling algorithm. VOQ has been used in chassis-based switch architecture. Extending it to the fabric provides better congestion handling compared to the DCQCN measures.

Topologies

In a data center, Clos is the most commonly used topology. It exists as a three-stage leaf-spine fabric, five-stage leaf-spine-super spine fabric, and so on. Clos topology, also referred to as fat-tree topology, provides a non-blocking architecture that fits the data center requirements. However, in the case of high-performance computing clusters (HPC) clusters and AI clusters, where the cluster size is considerably large, the Clos topology with its multiple stages introduces latency with every additional hop. Topologies that reduce the number of hops, such as Dragonfly (DF and DF+) topology and Torus topology, may be considered.

Dragonfly Topology

Dragonfly is a hierarchical topology in which multiple blocks, or bricks (or groups, or pods), connect with each other in a full mesh. Any topology can be implemented within the block. This topology reduces the number of hops compared to the five-stage Clos topology with a super spine and reduces the link requirements for a full mesh within the block. Eventually, it reduces the diameter, latency, and cost of multi-stage networks.

These are some of the benefits of the Dragonfly topology:

- **Scalability:** The Dragonfly topology is designed to be modular and highly scalable. It can be expanded by adding more compute node groups, so it allows straightforward growth to accommodate larger clusters.
- **Low latency:** Dragonfly is known for its low latency. Communication between nodes within the same group is extremely fast and experiences minimal latency. This makes Dragonfly topology suitable for applications that require real-time or near-real-time communication.
- **Fault tolerance:** The modular nature of Dragonfly topology enables fault tolerance. If a particular group or network component fails, the network can still function, although with reduced capacity.

Also, adaptive routing can help dynamically adjust to network conditions to optimize the data traffic and reduce congestion, as discussed in Chapter 8, "IP Routing for AI/ML Fabrics."

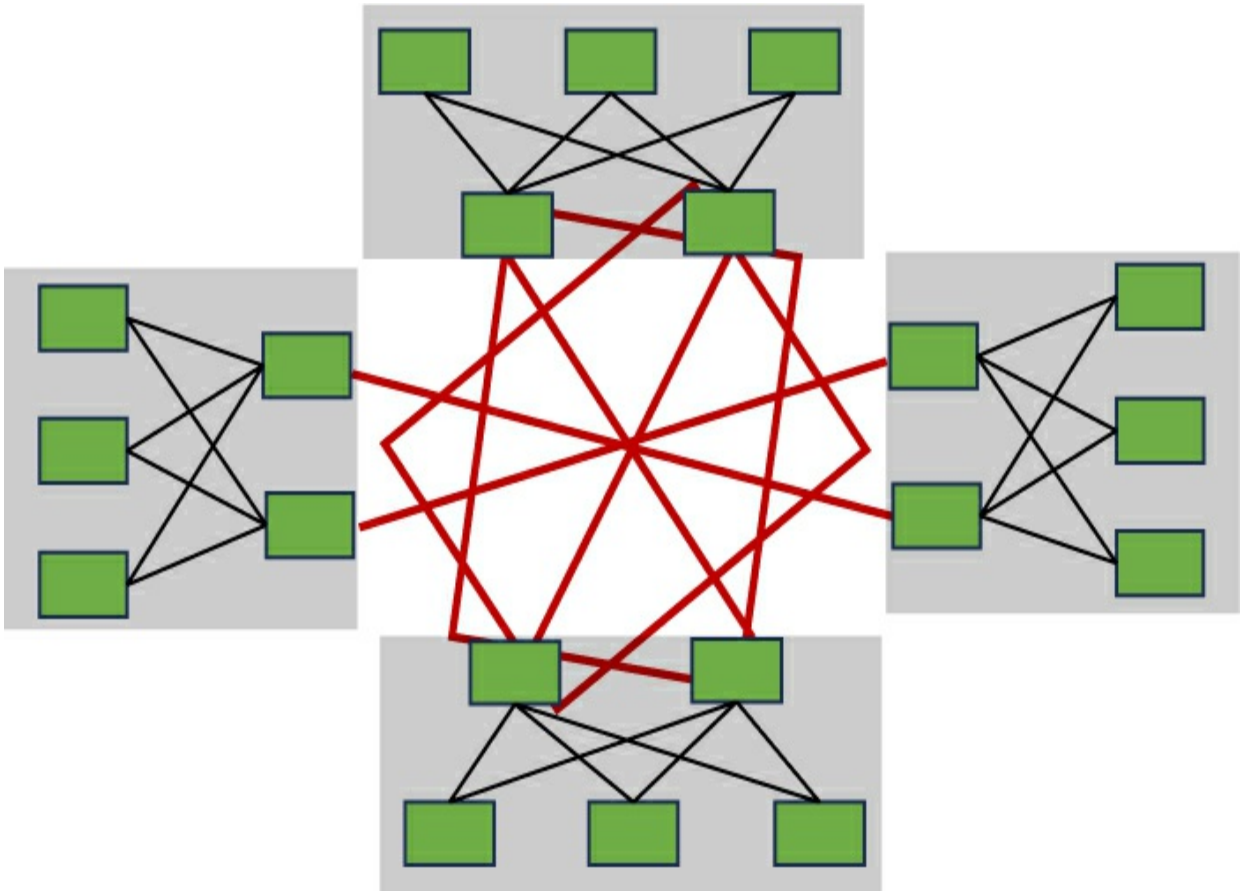


Figure 3-25 *Dragonfly topology*

Figure 3-25 illustrates Dragonfly topology with four blocks of three-stage Clos fabric connected with each other via a full mesh. In this topology, each spine in a block has mesh connections to all other spines, and each block is a full mesh to other blocks in two planes, as shown in Figure 3-26.

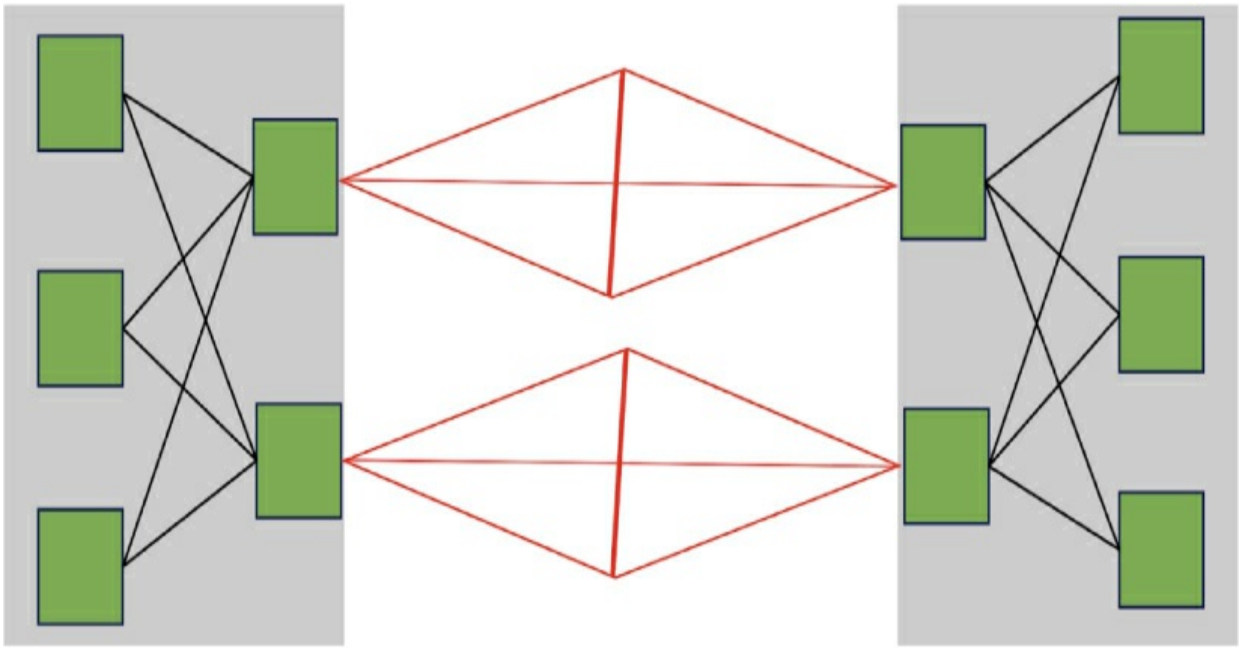


Figure 3-26 *Dragonfly topology planes view*

This topology is helpful for building large clusters with more GPUs for AI/ML fabrics.

Dragonfly supports multiple topologies within a group, or block. Of the different flavors of Dragonfly topologies, the following two topologies are worth mentioning:

- **Inter-group topology:** In this topology, which is also referred to as full-graph topology, all nodes within the group connect with each other in a mesh, as shown in Figure 3-27. The topology can be one dimensional or two dimensional.

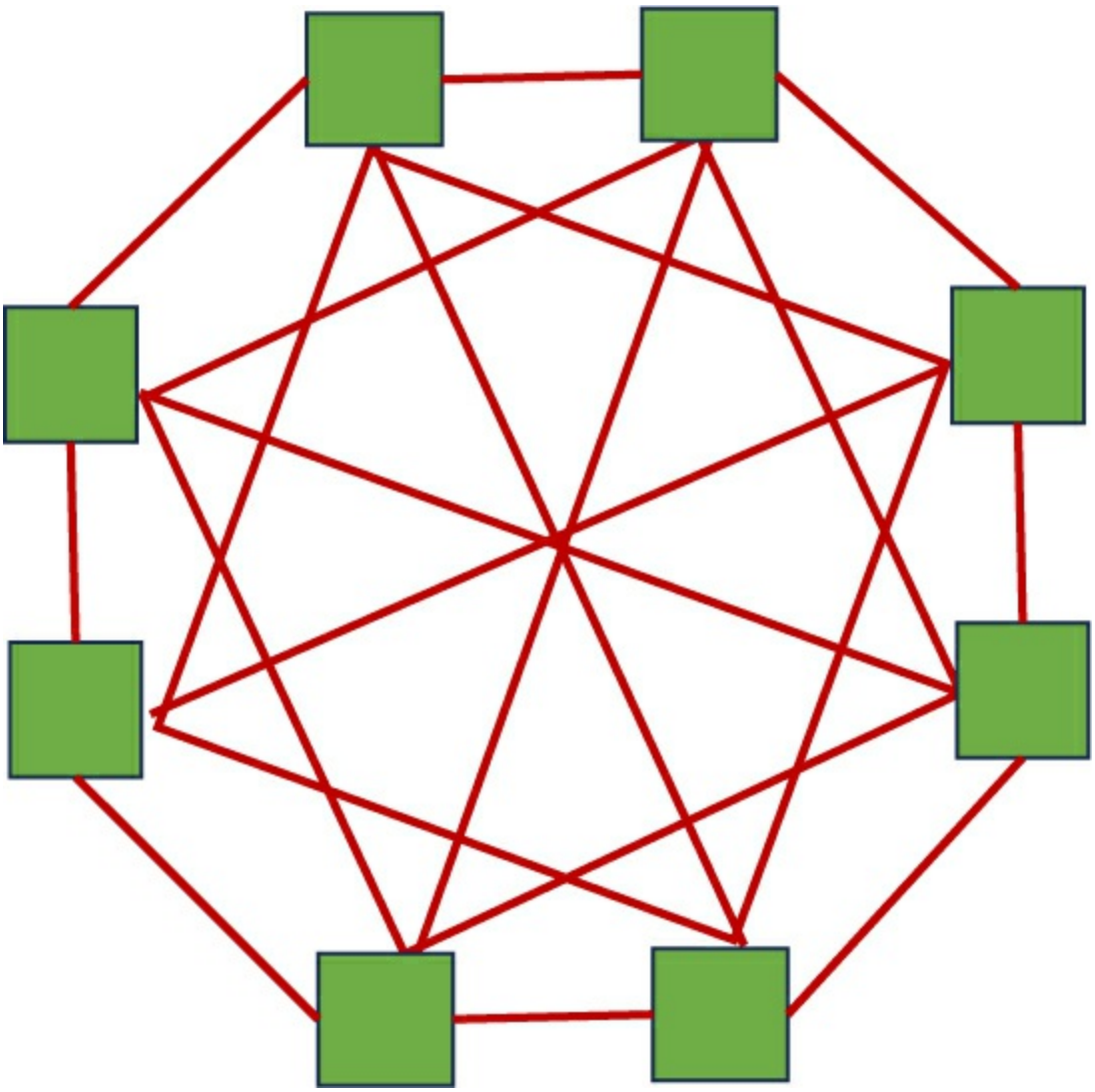


Figure 3-27 *Inter-group topology*

- **Clos topology:** This is a typical Clos fabric within a group, as shown in Figure 3-28. It is also referred to as Butterfly+ or full bipartite topology.

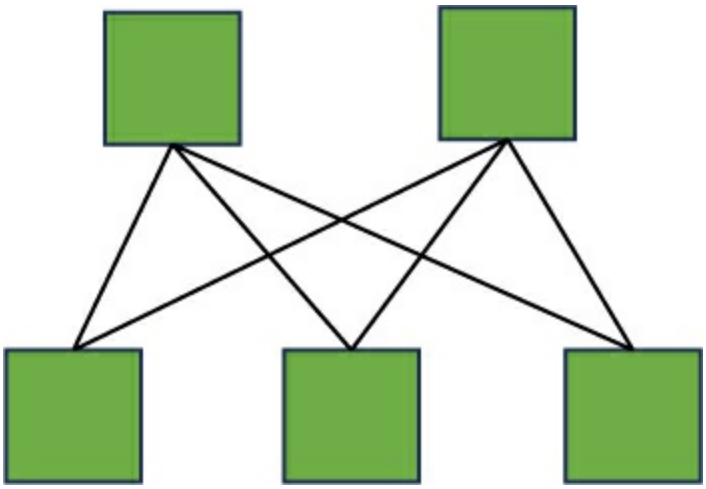


Figure 3-28 *Clos intra-group topology*

Torus Topology

A torus is a geometric shape that is generated by revolving a circle in a three-dimensional space. A doughnut is a good example of a torus shape. A torus network topology is interconnected, with nodes connecting to their neighboring nodes.

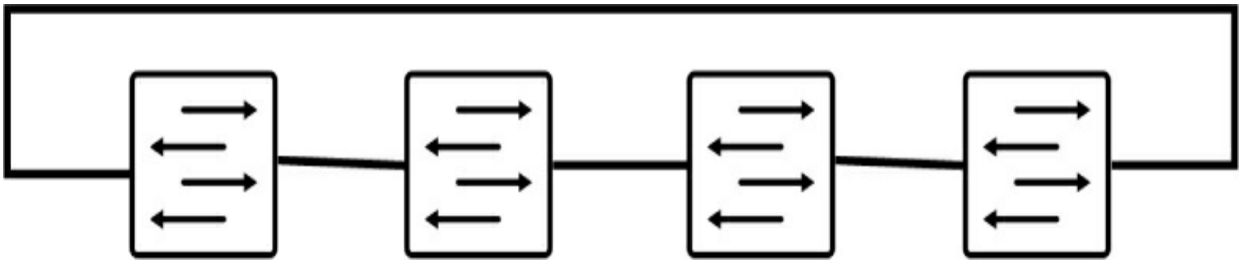


Figure 3-29 *One-dimensional torus topology*

Figure 3-29 illustrates a one-dimensional torus topology. In this topology, each node is connected to its neighbor on either side, as shown in Figure 3-30. The edge nodes are also connected to each other, and it looks like a ring topology. In one-dimensional Torus topology, each node has two connections.

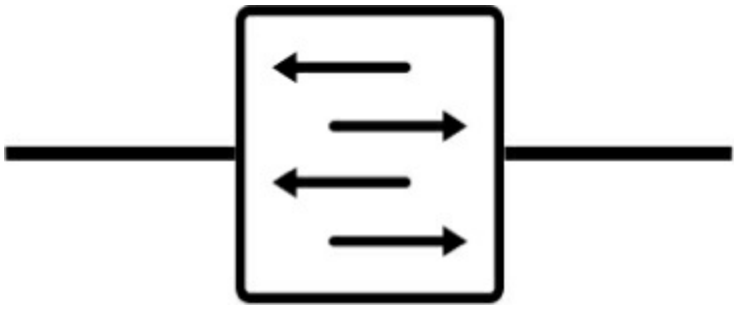


Figure 3-30 *One-dimensional node-level connectivity*

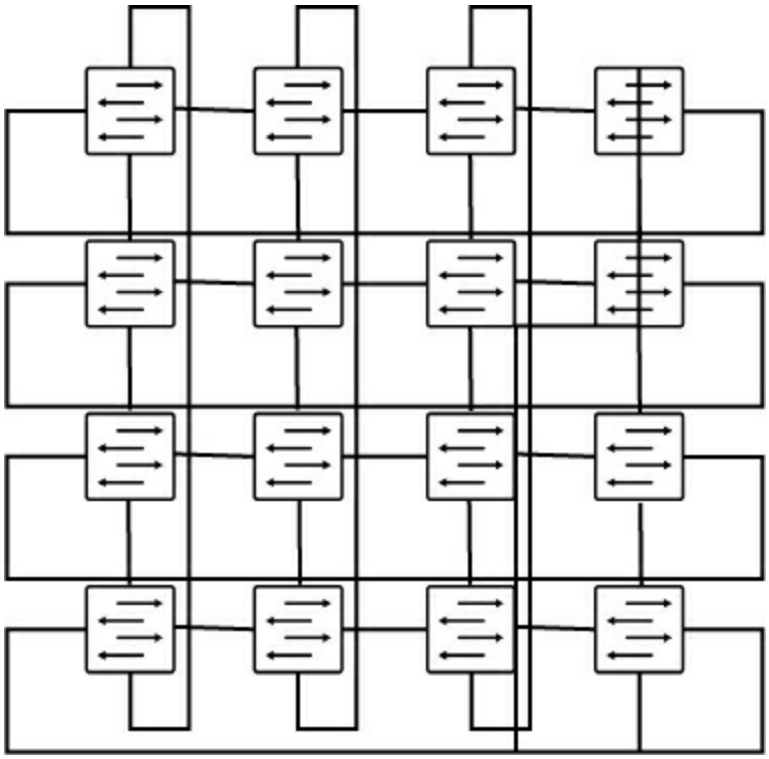


Figure 3-31 *Two-dimensional torus topology*

Figure 3-31 illustrates a two-dimensional torus topology. Each node is connected to its neighbor on either side, both horizontally and vertically, as shown in Figure 3-32. The edge nodes are also connected to each other in horizontal and vertical directions. In a two-dimensional Torus topology, each node has four connections.

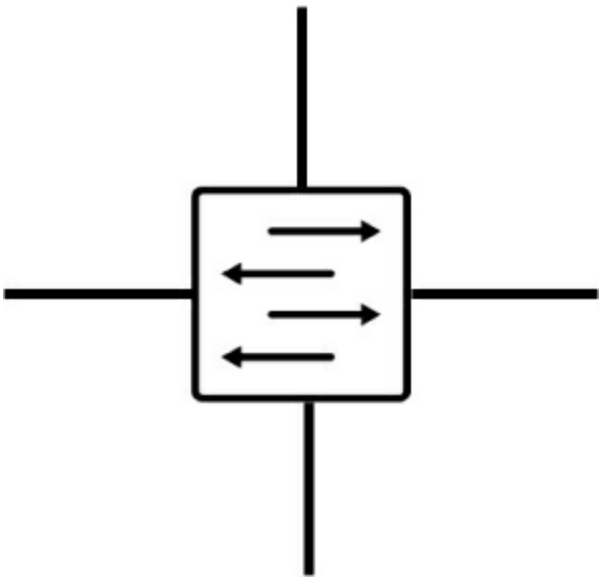


Figure 3-32 *Two-dimensional node-level connectivity*

Figure 3-33 illustrates a three-dimensional Torus topology.

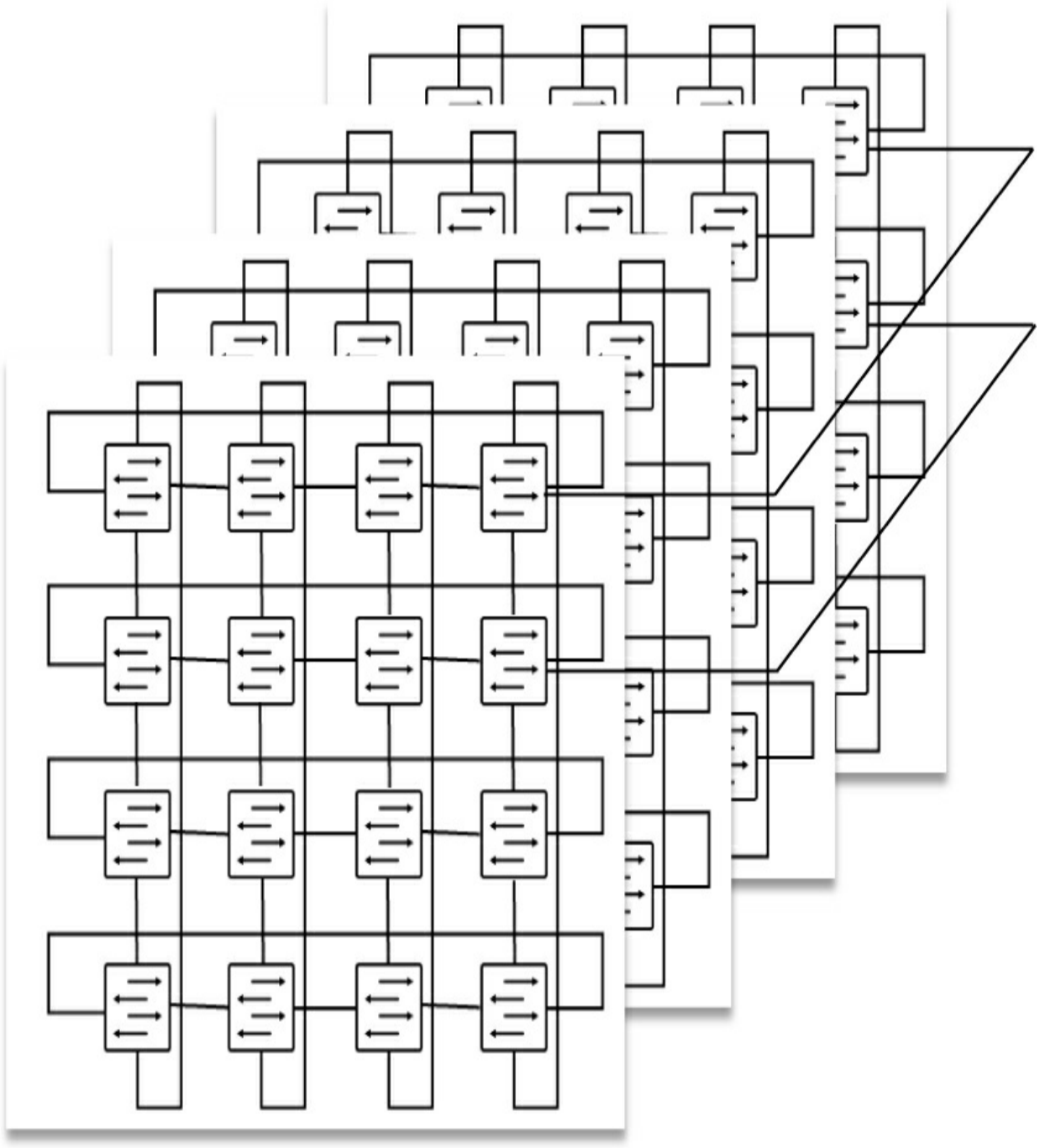


Figure 3-33 *Three-dimensional torus topology*

Each node is connected to its neighbor on either side along the x-axis, y-axis, and z-axis, as shown in Figure 3-34. The edge nodes are also connected to each other in the x-axis, y-axis, and z-axis. In the three-dimensional Torus topology, each node has six connections.

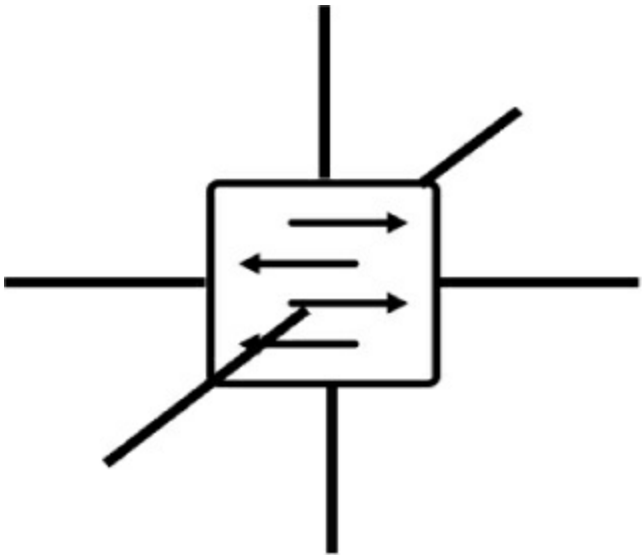


Figure 3-34 *Three-dimensional node-level connectivity*

As shown in Figure 3-35, it is possible to build a rail topology with a one-dimensional Torus topology. Instead of spine connectivity, the connections between the leafs enable inter-rail communication. For connecting multiple rails, a two-dimensional Torus or three-dimensional Torus topology can be used.

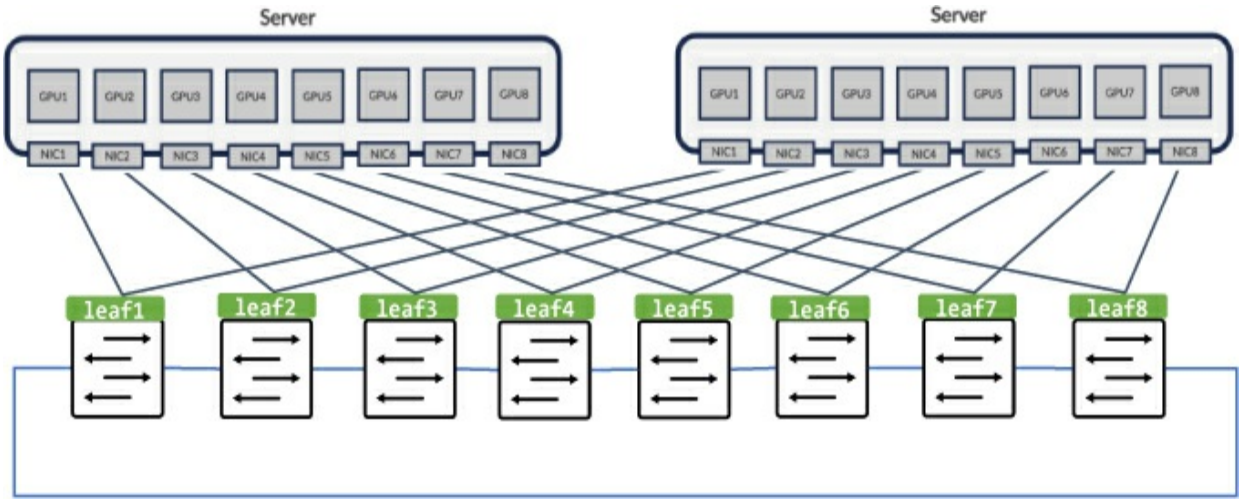


Figure 3-35 *Rail topology with a one-dimensional torus*

The torus topology offers many benefits:

- It is cost-effective at high scale due to the lower port requirement.
- It is relatively easy to scale out the topology.

- It offers the best performance when applications use nearest nodes.

Inference Data Center Architecture

Inference, as we have said, is a way of using a trained model and input data to get the required results. The deployment of inference data center architecture depends on the model size being used and the number of simultaneous users. There are a number of locations where inference models can be deployed:

- On a mobile device like a phone or a laptop for use by a single user
- In edge computing locations for multiple users, manufacturing units, retail, and so on
- In enterprise data centers
- In cloud data centers
- Co-located with training data centers

In most data centers, rail alignment is not required for inference data centers as the model can be run on a GPU or a small set of GPUs. Traffic in these cases may or may not use RDMA. The design follows Clos and typical data center design. However, the server side includes specialized GPUs and server connects.

Multi-node inference is not a common scenario and is required only in very special cases. Traffic in these cases is RDMA, and designers must determine whether to use the Clos or rail-based designs.

Summary

In this chapter, we have investigated different fabric design options and topologies for AI/ML workloads. Various options are being considered now, and eventually the most effective designs may be determined. In later chapters, we will discuss the routing options, optics, and thermal management for these topologies.

Test Your Knowledge

Chapter Review

The following questions are designed to test your understanding of the content covered in Chapter 3. Following the questions, answers are provided so you can verify your conclusions.

Questions

1. What are the main architectural stages in AI/ML workload processing, and how do they influence network design?
2. How do rail-optimized design (ROD) and rail-unified design (RUD) differ in terms of GPU-to-leaf connectivity, and what are the implications for scalability and fault tolerance?
3. What are the key considerations and trade-offs in rack design (top of rack, middle of row, end of row) for AI/ML data centers?
4. How does the Clos (fat-tree) topology enable scalable, non-blocking AI/ML fabrics, and what are the challenges at extreme scale (for example, 32,000+ GPUs)?
5. What are the benefits and limitations of alternative topologies like Dragonfly and Torus for AI/ML clusters?
6. How do scheduled fabric architectures (for example, cell-based switching, VOQ) improve congestion management and link utilization in AI/ML data centers?
7. What are the power and cooling implications of high-density AI/ML racks, and how do they influence network and facility design?
8. How do deterministic path forwarding and traffic engineering support performance and reliability in large-scale AI/ML fabrics?

Answers

1. AI/ML workload processing typically follows three main stages:

- Data gathering and preprocessing: Data is collected from diverse sources and cleaned, labeled, and tagged. This stage requires high-throughput storage networks and efficient data pipelines to ensure that large data sets can be ingested and prepared without bottlenecks.
- Model selection and training: The curated data is used to train models, often requiring clusters of GPUs interconnected with high-bandwidth, low-latency fabrics. The network must support massive east–west traffic, parallel data transfers, and collective communication operations.
- Deployment and monitoring: Trained models are deployed for inference, and their performance is monitored. This stage may involve both backend (training) and frontend (inference) networks, each with different latency, bandwidth, and reliability requirements.

2. These are the differences between ROD and RUD:

- ROD: Each GPU in a server is connected to a separate leaf switch, forming “rails.” This design minimizes intra-rail latency and maximizes bandwidth, as each GPU has a dedicated path to the network. It allows for efficient scaling by adding more rails (leaf switches) and supports high GPU counts per cluster. Faults are isolated to individual rails, limiting the impact of a switch failure.
- RUD: Multiple GPUs from a server connect to the same leaf switch. This simplifies cabling and can reduce costs, but it increases the risk that a single leaf switch failure will isolate all GPUs in a server. Scaling is easier in terms of cabling but less optimal for performance and fault isolation compared to ROD.

ROD is preferred for large-scale, high-performance clusters where minimizing latency and maximizing parallelism are critical. RUD may be chosen for smaller deployments or where cost and cabling simplicity are prioritized and when larger chassis-based systems are used (for example, a higher-end multi-linecard Ethernet switch instead of a 1RU ToR).

3. These are the key trade-offs:

- Top of rack (ToR): Switches are placed at the top of each rack, minimizing intra-rack cable lengths and simplifying management.

However, this design increases per-rack power and cooling requirements.

- Middle of row (MoR): Switches are centralized in the middle of a row, reducing the number of switches but increasing cable lengths and complexity.
- End of row (EoR): Switches are placed at the end of a row, further centralizing switching but requiring the longest cables and careful planning for airflow and power.

ToR is best for high-density, high-performance clusters where minimizing latency and maximizing manageability are key. MoR and EoR can reduce switch count and cost but may complicate cabling and cooling. The choice depends on cluster size, density, and operational priorities.

4. The Clos topology uses multiple stages (leaf, spine, super spine) to create a non-blocking, highly parallel network. Each leaf connects to every spine, ensuring multiple equal-cost paths between any two endpoints. Clos can scale to tens of thousands of endpoints with the addition of more stages and higher-radix switches. At extreme scale, challenges include managing oversubscription, cabling complexity, and power/cooling and maintaining low-latency paths. As the number of GPUs increases, the number of required switch ports and interconnects grows rapidly. Multi-stage Clos (three-stage, five-stage, seven-stage) and chassis-based spines/super spines are used to manage this, but careful planning is needed to avoid bottlenecks and ensure efficient load balancing.
5. Dragonfly connects groups of switches in a mesh, reducing the number of hops and the network diameter. It offers lower latency and better scalability for certain traffic patterns but requires more complex routing and fault management.

   Torus connects nodes in a ring (1D), grid (2D), or cube (3D), providing multiple paths and high bisection bandwidth for nearest-neighbor communication. It is cost-effective and easy to scale but can suffer from higher average latency and less flexibility for arbitrary traffic patterns.

Both topologies can complicate routing, require specialized hardware/software support, and may not be as flexible as Clos for mixed or unpredictable workloads.

6. Scheduled fabric splits packets into fixed-size cells, uses virtual output queueing (VOQ) to prevent head-of-line blocking, and schedules cell transmission across the fabric. This approach allows for fine-grained congestion management, efficient spraying of traffic across multiple links, and improved fairness.

   The key benefits of scheduled fabric are reducing latency, maximizing link utilization, and preventing congestion hot spots. Scheduled fabric is particularly effective for bursty, synchronized AI/ML workloads, where traditional packet-based switching may struggle. Scheduled fabric requires advanced switch hardware and careful configuration, and it may introduce additional complexity in troubleshooting and monitoring, as well as higher costs compared to a distributed lossless Ethernet fabric with shallow buffers.

7. Modern AI/ML servers (for example, DGX H100) can require up to 15 kW per server, with racks exceeding 60 kW. This necessitates advanced power distribution, redundant supplies, and high-capacity cooling (liquid, immersion, or advanced airflow). High-density racks may require distributed switches (for example, ToR) to minimize cable lengths and manage heat. Facility design must ensure adequate power delivery, cooling capacity, and airflow management to prevent hotspots and ensure reliability. Power and cooling are primary constraints in scaling AI/ML clusters, influencing rack placement, switch location, and overall data center architecture.

8. Deterministic path forwarding ensures that specific flows (for example, between GPUs in the same rail) follow predictable, optimized paths, reducing contention and improving performance. This is achieved through careful network configuration, use of deterministic routing protocols, and sometimes explicit path pinning.

   Traffic engineering involves dynamically adjusting routing and load balancing based on real-time network conditions, workload requirements, and failure scenarios. Techniques include ECMP, flowlet-based balancing, and policy-based routing. Such an approach

improves reliability, reduces tail latency, and ensures that critical AI/ML jobs meet performance SLAs even as the network scales and workloads fluctuate.

References

https://www.advancedclustering.com/wp-content/uploads/2022/03/gtc22-whitepaper-hopper.pdf

https://images.nvidia.com/aem-dam/en-zz/Solutions/data-center/nvidia-ampere-architecture-whitepaper.pdf

https://en.wikipedia.org/wiki/Torus_interconnect

Chapter 4. Optics and Cables Management [This content is currently in development.]

This content is currently in development.

Chapter 5. Thermal and Power Efficiency Considerations [This content is currently in development.]

This content is currently in development.

Part 3: AI/ML Data Center Technology Requirements

Chapter 6. Efficient Load Balancing [This content is currently in development.]

This content is currently in development.

Chapter 7. RoCEv2 Transport and Congestion Management

RDMA over Converged Ethernet (RoCEv2) is the most popular transport protocol used to synchronize data chunks between application buffers on distributed GPU servers of an AI/ML cluster. It uses UDP, which does not require maintaining state on the GPU NIC, as the transport protocol. Because RoCEv2 does not engage the server CPU, it scales better than any other TCP-based data transport, such as Non-Volatile Memory Express over TCP (NVMe/TCP), a protocol that is often used for traditional storage data networks. However, despite offering better parallel session scalability, RoCEv2 has entropy characteristics that present networking challenges. The traffic originating from GPUs may cause either momentary or persistent traffic congestion inside the network, resulting in slower data synchronization between GPUs.

In the previous chapter, we discussed various load-balancing techniques to better utilize the fabric. However, even with the best load-balancing mechanisms, congestion may still occur, especially in the case of local switching at a leaf, where load-balancing mechanisms are not prevalent. Also, sudden bursts in a flow can result in congestion in an otherwise well-balanced fabric. Later in this chapter, we will discuss the various congestion management techniques for RoCEv2.

Congestion Points

Let’s consider congestion points in detail. With no oversubscription in the fabric and line rate traffic from servers, there is a high chance of congestion at various links in the network:

- **Local leaf link congestion:** Assume that two servers and a storage device are connected to LeafA, as shown in Figure 7-1. Both servers are sending line rate traffic toward the locally connected storage server on the same link, resulting in congestion.

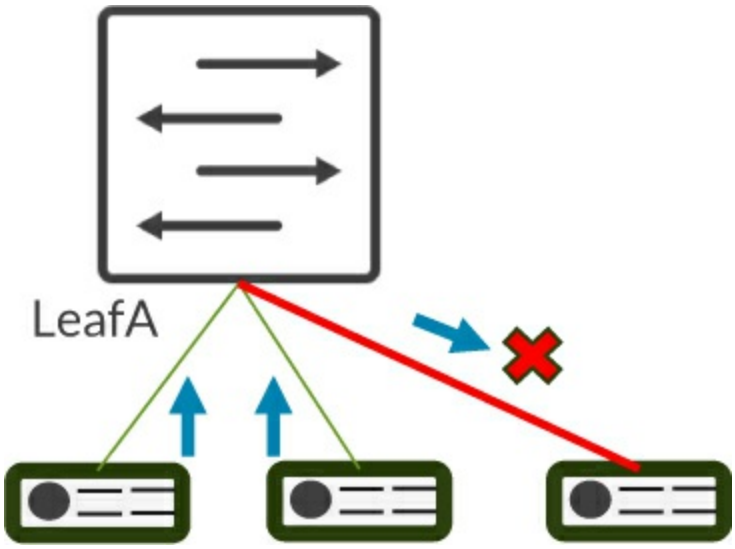


Figure 7-1 *Incast congestion toward south-facing links*

- **Leaf-to-spine link congestion:** Assume that two servers are connected to LeafA, as shown in Figure 7-2. Both servers are sending line rate traffic toward SpineA. The traffic is load balanced into the same link from LeafA toward SpineA, which can cause congestion in the leaf-to-spine link.

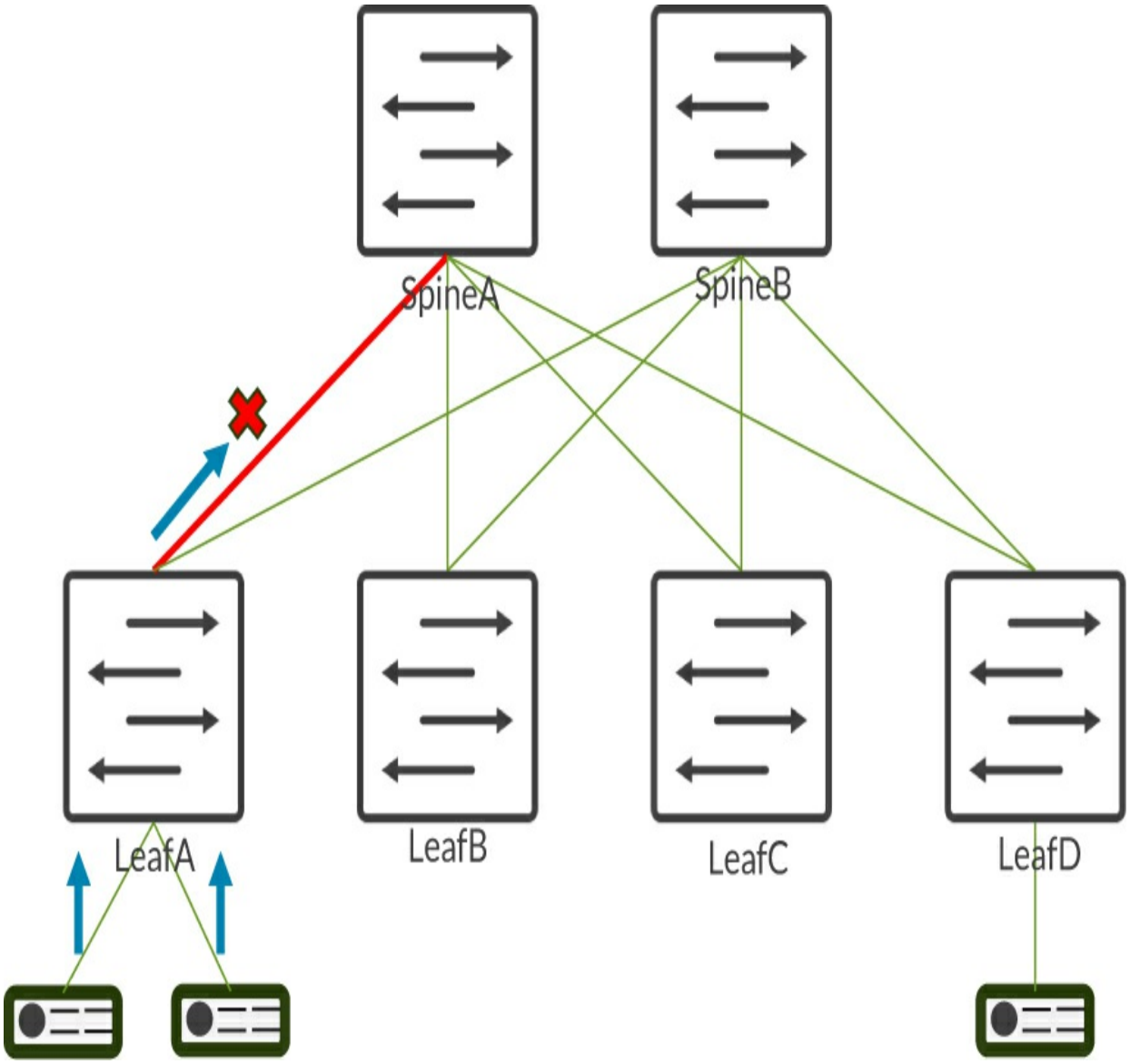


Figure 7-2 *Incast congestion at an ingress leaf*

- **Spine-to-leaf link congestion:** Assume that a server is connected to LeafA and another server is connected to LeafB, as shown in Figure 7-3. Both servers are sending line rate traffic. Traffic from LeafA and LeafB reaches SpineA without any congestion. However, when traffic is load balanced into the same link from SpineA toward LeafD, it can result in congestion in the spine-to-leaf link.

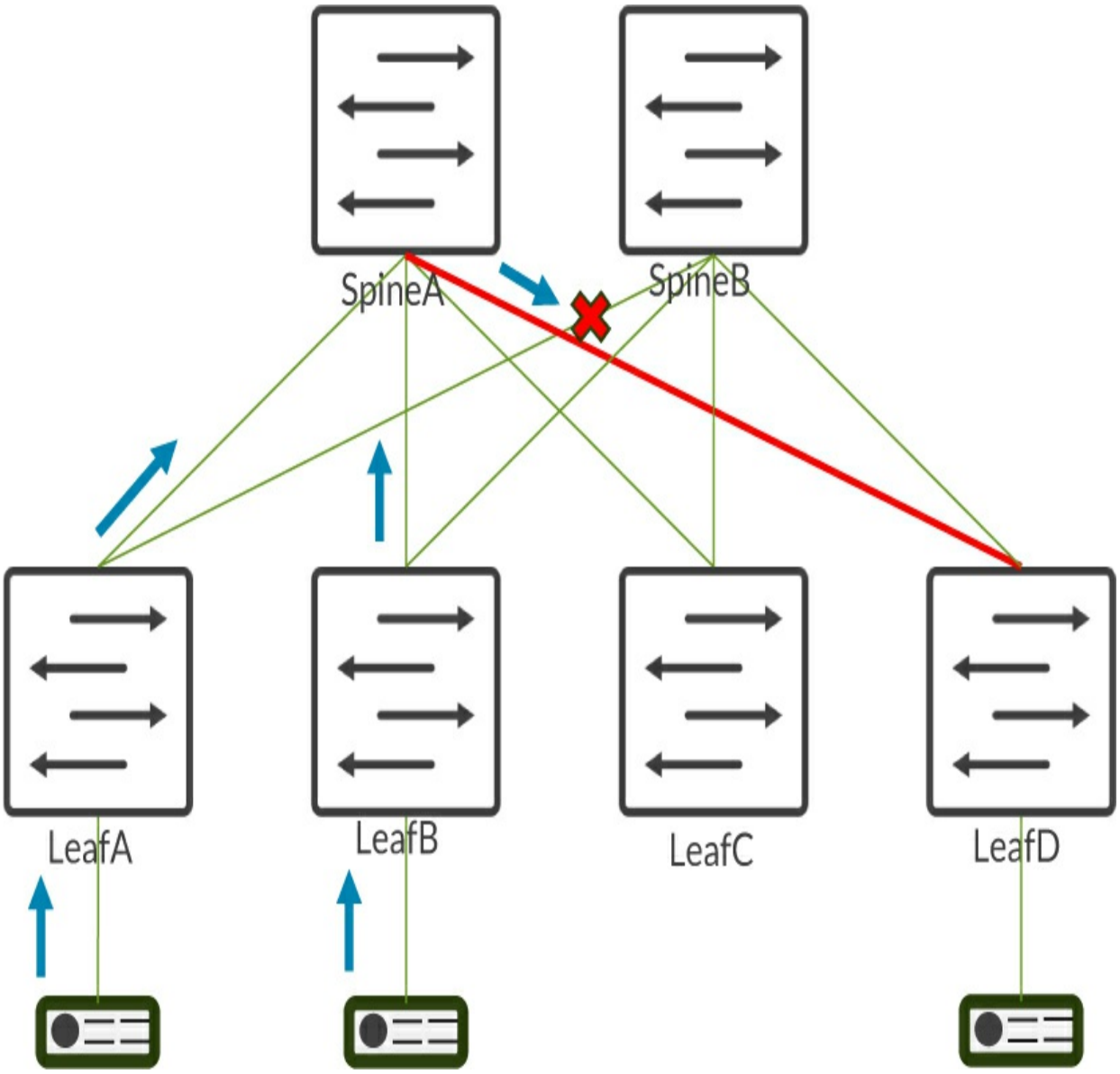


Figure 7-3 *Incast congestion at a transit spine*

- **Leaf-to-server link congestion:** Assume that a server is connected to LeafA and another server is connected to LeafB, as shown in Figure 7-4. Both servers are sending line rate traffic. The traffic is distributed between two spines, SpineA and SpineB, and it will reach LeafD without any congestion. However, if traffic volume is greater than the bandwidth of the link between LeafD and the server attached to it, there may be congestion in the leaf-to-server link.

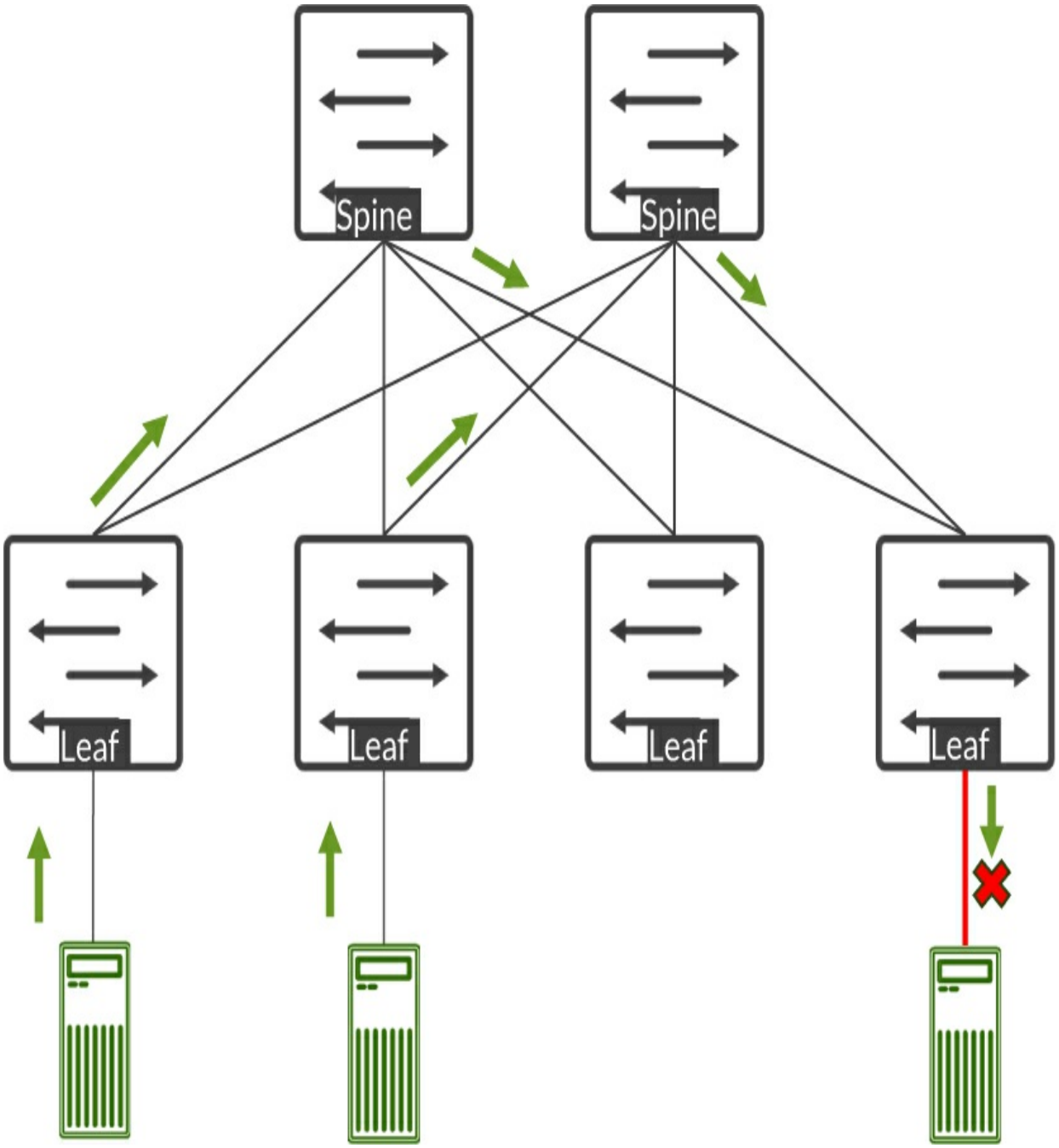


Figure 7-4 *Incast congestion at an egress leaf*

- **Spine-to-super spine link congestion:** Assume that a server is connected to LeafA and another server is connected to LeafB, as shown in Figure 7-5. Both servers are sending line rate traffic. Traffic from LeafA and LeafB reaches SpineA without any congestion. However, when traffic is load balanced into the same link from SpineA toward SuperSpineA, it can result in congestion in the spine-to-super spine link.

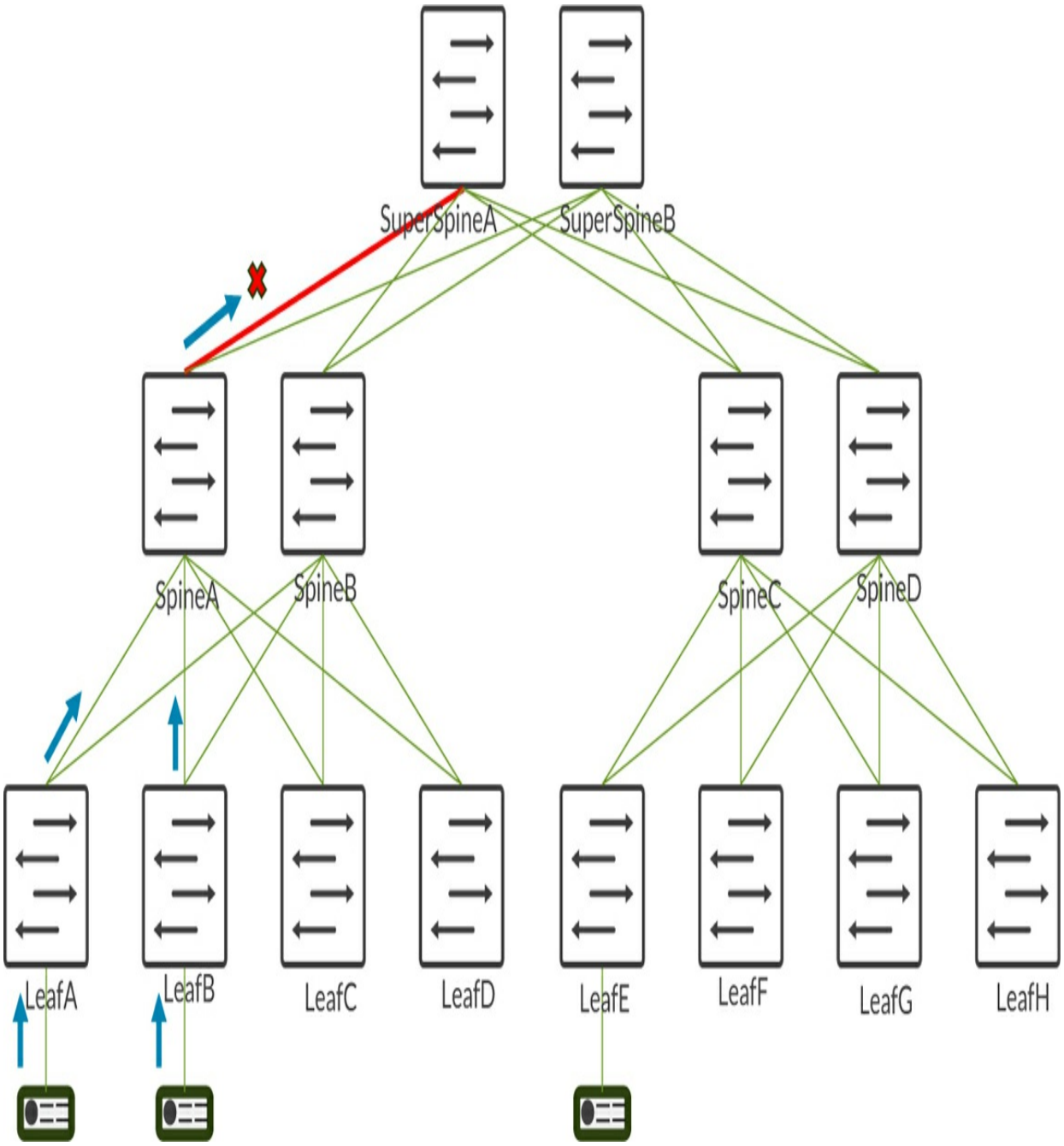


Figure 7-5 *Incast congestion at the spine-to-super spine link*

- **Super spine-to-spine link congestion:** Assume that a server is connected to LeafA and another server is connected to LeafB, as shown in Figure 7-6. Both servers are sending line rate traffic. Traffic from LeafA and LeafB reaches SpineA and SpineB without any congestion. Also, traffic is load balanced into the links toward SuperSpineA without congestion. However, when traffic is load balanced into the same link

from SuperSpineA to SpineC, it can result in congestion in the super spine-to-spine link.

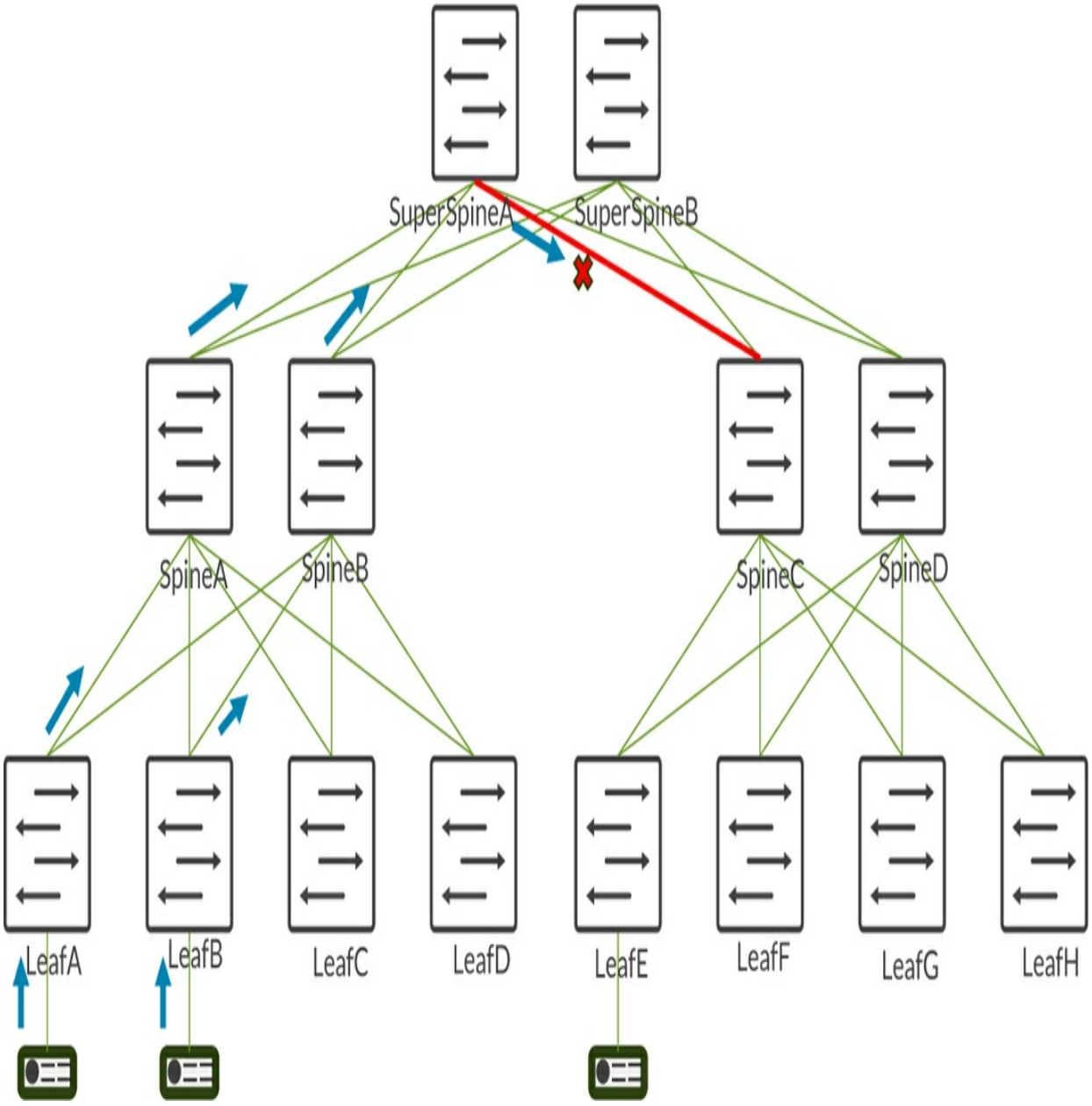


Figure 7-6 *Incast congestion at the super spine-to-spine link*

Storage networks that use RoCEv2 must be lossless. However, Ethernet is lossy. It supports two transport protocols: TCP and UDP. TCP is reliable because it uses acknowledgement to detect packet drops and retransmits lost packets. It also uses a windowing mechanism to reduce the drops in the network. UDP, on the other hand, is unreliable and more prone to losses.

Therefore, Ethernet requires implementation of congestion management techniques, as we'll discuss next.

Explicit Congestion Notification (ECN)

Explicit Congestion Notification (ECN) enables end-to-end congestion notification between endpoints—that is, between a sender and a receiver. ECN must be enabled on both of the endpoints as well as on intermediate devices between them. The ECN threshold needs to be configured on the transit switches. Any device in the transmission path that does not have ECN enabled breaks the end-to-end ECN functionality.

ECN works with ECN bit marking and congestion notification packets (CNPs). In an IP header, there are 8 bits for Differentiated Services Code Point (DSCP). The first 6 bits are for Quality of Service (QoS) or Class of Service (CoS) marking. The last 2 bits are used for ECN marking. The first ECN bit signifies ECN-capable transport (ECT), identifying whether the transport is ECN capable. The second bit signifies congestion experienced (CE), identifying congestion on the link. Table 7-1 explains the ECN bit values.

Table 7-1 *ECN Bits*

ECT Bit	CE Bit	Description
0	0	The transport is not ECN capable. In the event of congestion, the packet will be dropped instead of being marked with the ECN bits.
0	1	The transport is ECN capable. This is set by the receiver toward the sender at the time the CNP is sent. ECN 01 and 10 were created as two different values for some experimental purpose. From a networking point of view, they are treated the same.
1	0	The transport ECN capability is enabled. In the event of congestion, the packet will be marked ECN instead of being dropped. This bit is set by the sender toward the receiver. It can be set by the receiver toward the sender at the time the CNP is sent.
1	1	Congestion has been experienced on the link. In the event of congestion, the router will mark the packet with the ECN bits and send it to the receiver.

The CNP is generated by the receiver or the destination server. It is a RoCEv2 frame with ECN bits set to 01. The IB BTH opcode is 129 (10000001). It contains the destination queue pair information, which helps the sender identify the flow under congestion and reduce the traffic rate for that specific flow.

Figure 7-7 illustrates the flow of packets between servers attached to LeafA and LeafD:

1. The server attached to LeafA initiates traffic flow.
2. LeafA forwards the traffic to SpineA.
3. Congestion is detected on the link between SpineA and LeafD. As a result, SpineA marks packets with the ECN bit set to 11.
4. When the ECN-marked packet reaches the receiver, it understands there is congestion along the path. It generates a CNP with the ECN bit set to 01 and sends it to the sender.
5. The sender receives the CNP, understands there is flow congestion, and reduces the rate of traffic.

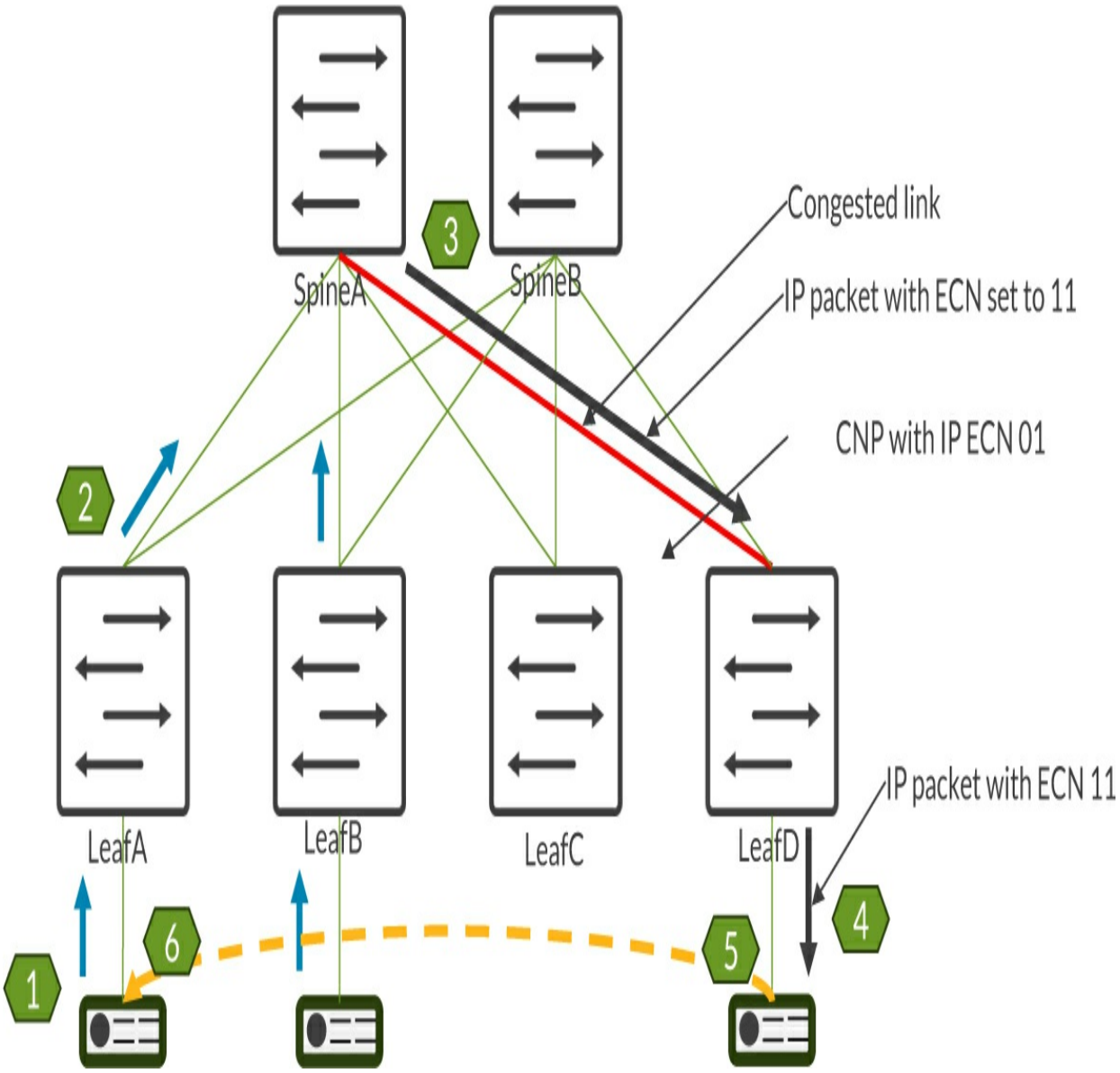


Figure 7-7 *ECN behavior*

Figure 7-8 shows the message flow between sender and receiver with ECN enabled.

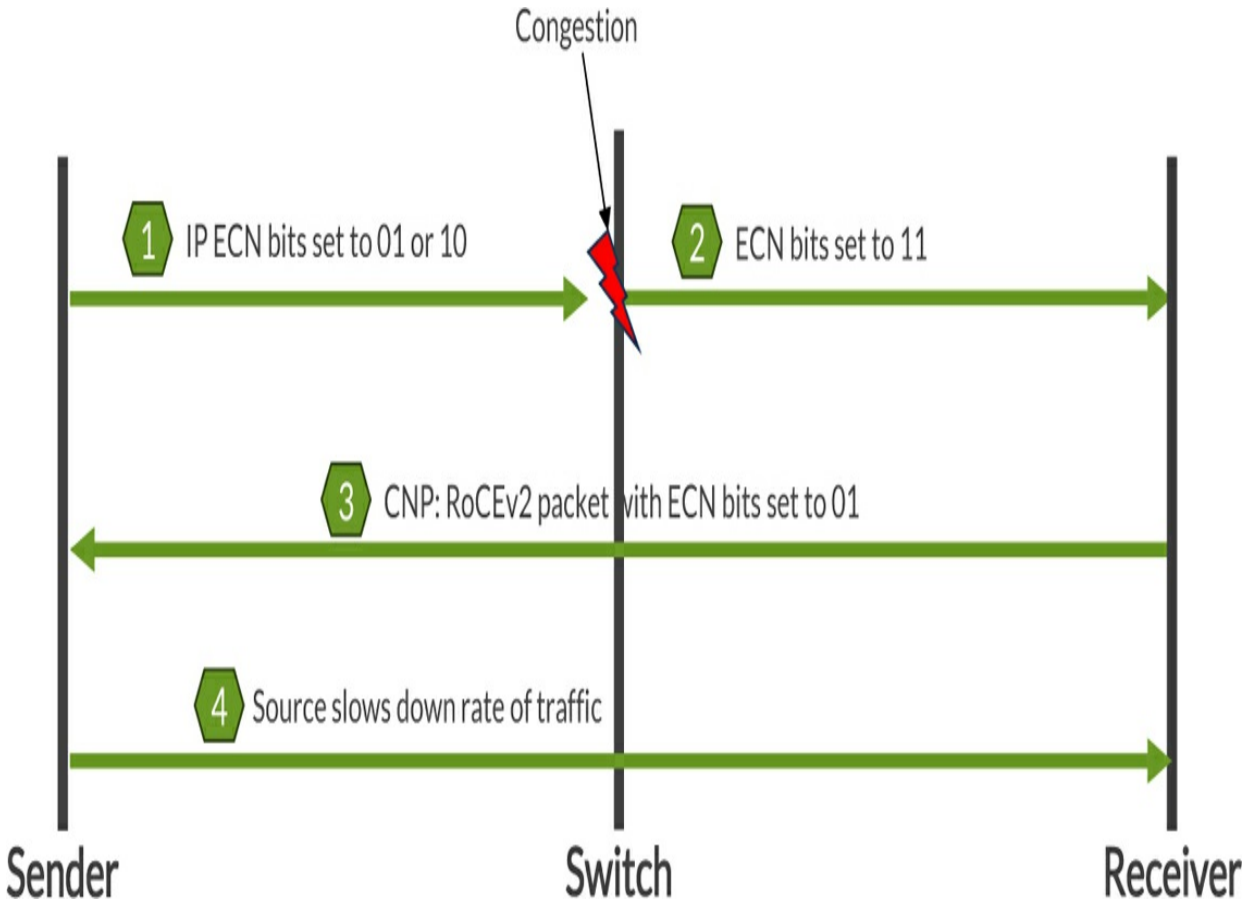


Figure 7-8 *ECN-enabled data transfer*

Congestion is detected based on the queue usage and an ECN threshold. Figure 7-9 illustrates the ECN threshold in a buffer. The ECN threshold is a static but configurable value. When the buffer utilization crosses this threshold, the switch marks ECN bits in the packet. The path of the packet remains the same. Beyond the threshold, the switch may apply weighted random early detection (WRED) and drop some packets, based on the buffer utilization. Operators are exploring ways to monitor ECN statistics and dynamically modify the threshold. It is possible for a number of flows to share the same queue, and ECN is evolving to support such scenarios.

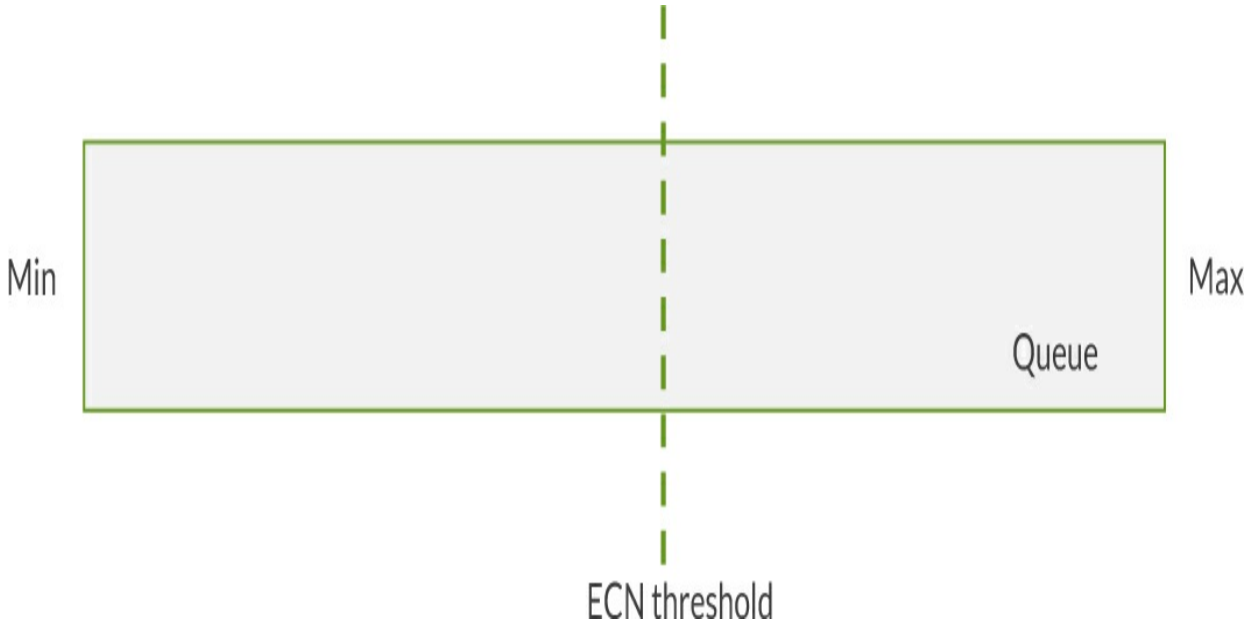


Figure 7-9 *ECN threshold*

While ECN is a popular congestion control mechanism, a problem with ECN is the time it takes to inform the source of the congestion. As discussed earlier, first the ECN-marked packet reaches the receiver, and then the receiver informs the source by sending a CNP. In the meantime, if the traffic continues to increase, packet drops may occur, which is not acceptable for AI/ML fabrics. Therefore, a queue with ECN enabled is referred to as a *lossy queue*.

Priority Flow Control (PFC)

Routers and switches use Priority Flow Control (PFC) in the transit path to inform the upstream hop about congestion for a traffic class. PFC uses pause frames, also referred to as XOFF frames, to control congestion. A network device generates a pause frame and sends it upstream, toward the source. The frame is generated for the class of traffic experiencing congestion such that the upstream devices stop forwarding traffic for the entire class of traffic—and not just for a flow. A queue on which PFC is configured is referred to as a *lossless queue*.

Figure 7-10 illustrates a PFC frame, which is a Layer 2 frame with the destination address set to unknown unicast. The source address is the address of the switch that generated the frame. Ether Type is set to 0x8808 to indicate

a PFC frame. Priority Control Vector or class-enabled bits are used to specify which class or classes the PFC pause is requested for. Time is a 16-octet field with 2 octets for each class of traffic. It indicates the time duration for which traffic to the queue is paused. The value 0 indicates un-pause for the traffic class.

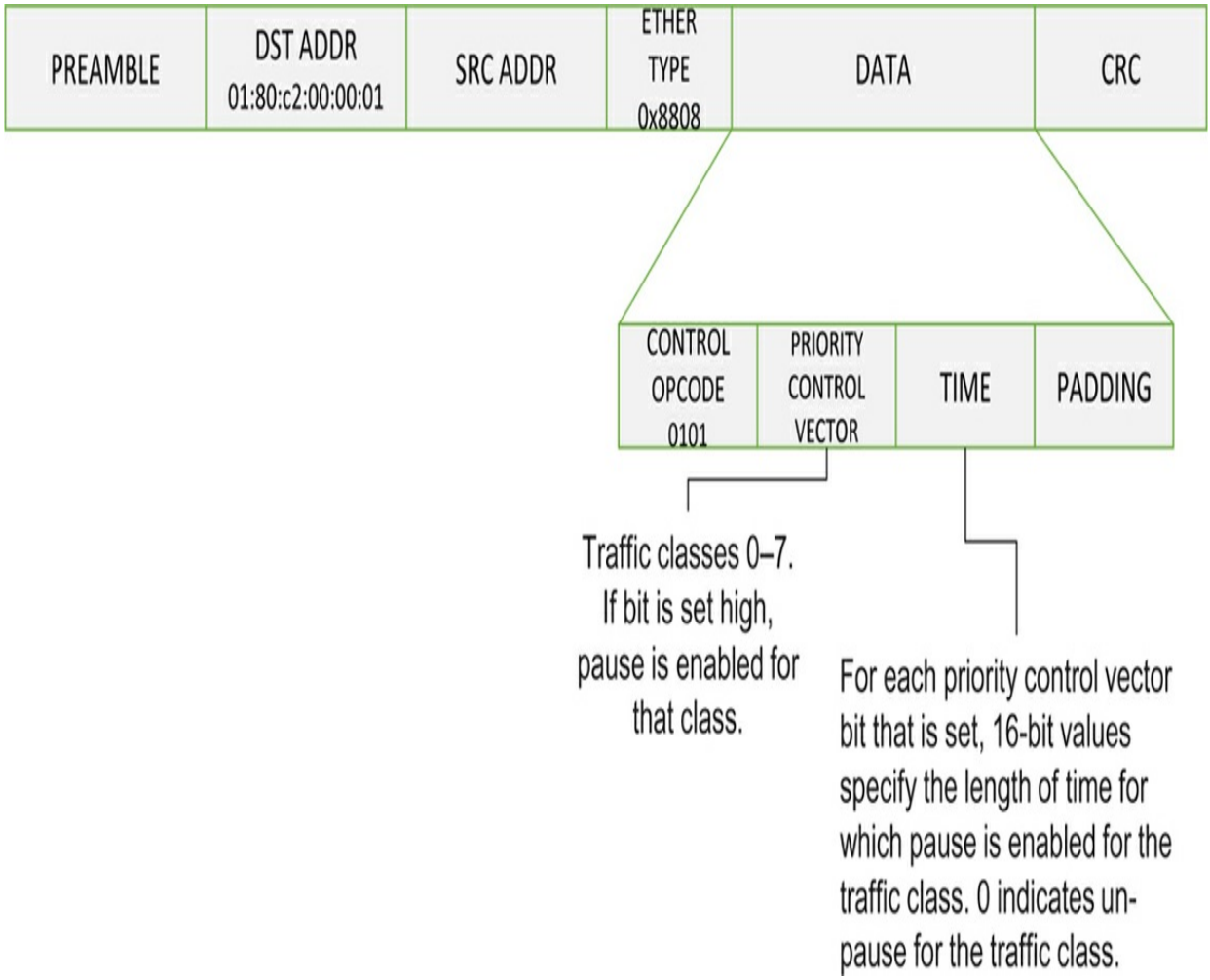


Figure 7-10 *PFC frame*

There are two buffer thresholds used with PFC: XOFF (Transmit Off) and XON (Transmit On). During congestion, the buffer starts to fill up due to congestion on the egress port. When buffer size crosses the XOFF threshold, a PFC pause frame is sent upstream to pause traffic associated with the class of traffic. Figure 7-11 illustrates the PFC pause, or XOFF, frame generated for the C3 class of traffic. The timer field in the frame is set to 65535 microseconds. If no further PFC pause frames are received, transmit will be

turned on after this period. If new PFC pause frames are sent upstream, the timer will be reset to a new value.

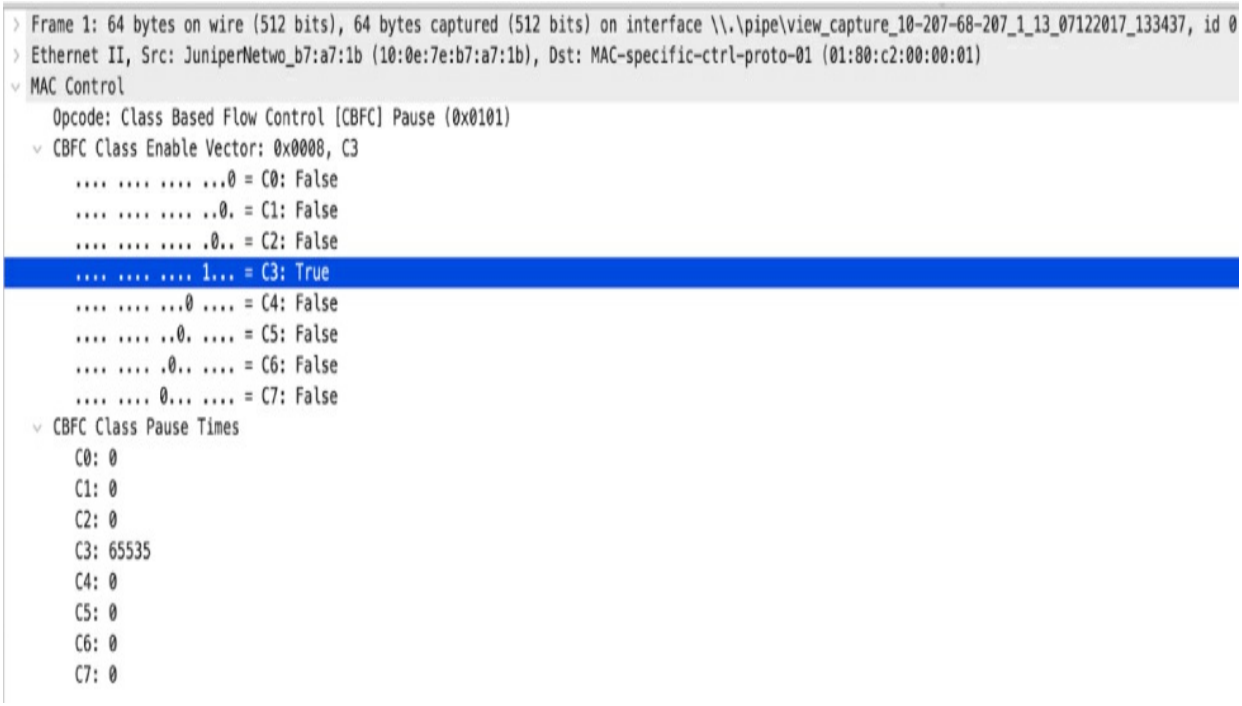


Figure 7-11 *PFC pause, or XOFF, frame*

When congestion is mitigated and buffer utilization falls below the XON threshold, an XON frame is generated and sent to the upstream switch. The switch starts the data transmission. Figure 7-12 illustrates a PFC XON frame generated for the C3 class of traffic. The frame also has the timer set to 0 microseconds, which indicates that data transmission can be resumed.

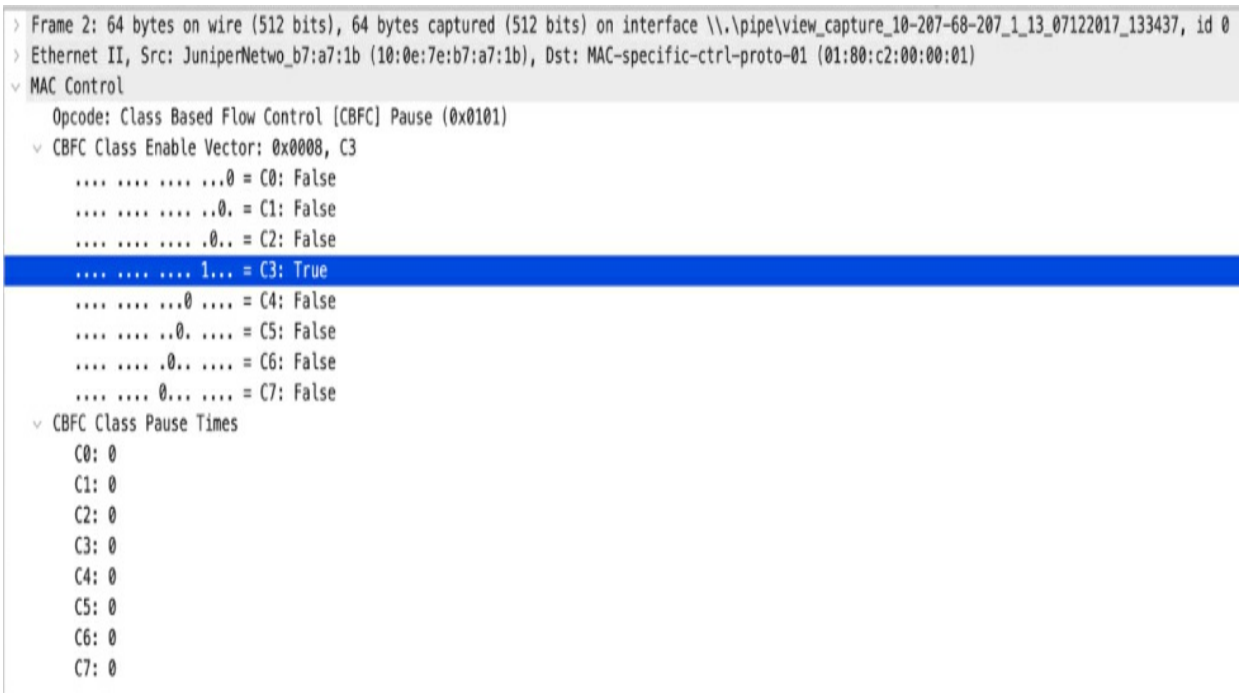


Figure 7-12 *PFC XON frame*

Figure 7-13 illustrates the flow of packets with PFC implemented:

1. The server attached to LeafA initiates the packet flow.
2. LeafA forwards the packets towards SpineA.
3. Congestion is detected in the link from SpineA to LeafD, based on the queue usage and PFC threshold.
4. SpineA sends a PFC pause, or XOFF, frame for the queue under congestion. For each packet that crosses the XOFF threshold, an XOFF frame is generated.
5. LeafA receives the PFC pause frame and pauses the class of traffic in the congested queue for the duration set for the timer field in the pause frame. Also, LeafA sends the PFC pause frame downstream toward the server.
6. Upon receiving the PFC pause frame, the server also pauses the flow for the duration set for the timer field in the pause frame.

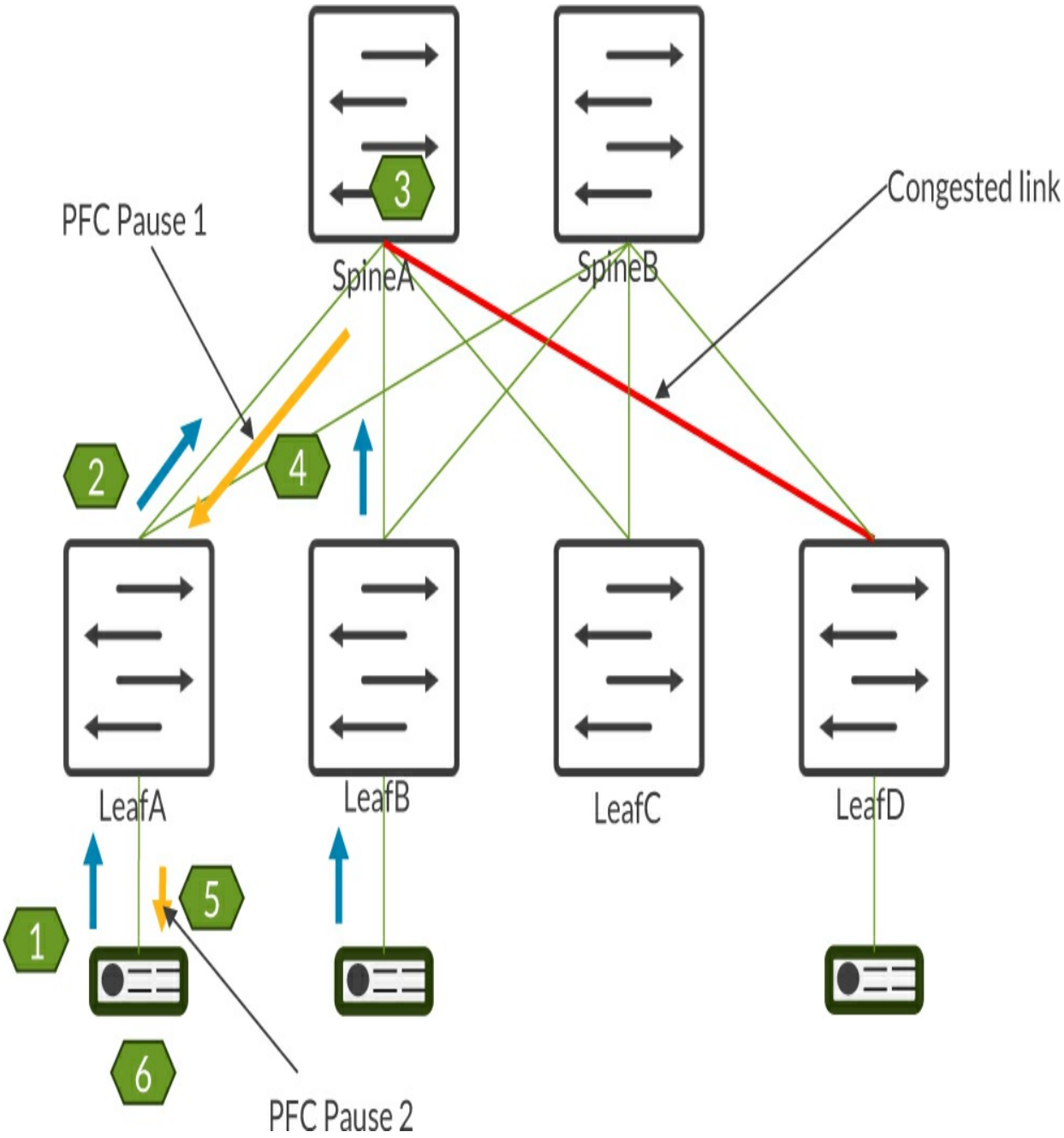


Figure 7-13 *PFC behavior*

Figure 7-14 illustrates threshold levels for XOFF and XON. The thresholds must be configured appropriately for an environment. XOFF should not be set to the maximum buffer size, or traffic drops may occur before congestion control activates. XOFF also should not be set too low, or PFC might pause the traffic for a class at lower queue utilization. The XON threshold should not be set to the minimum buffer size, as this setting may result in delayed

traffic transmission once congestion is mitigated. It is a good idea to monitor the PFC statistics and adjust the thresholds. The XOFF value can either be configured directly or controlled via the alpha configuration, based on the chipsets.

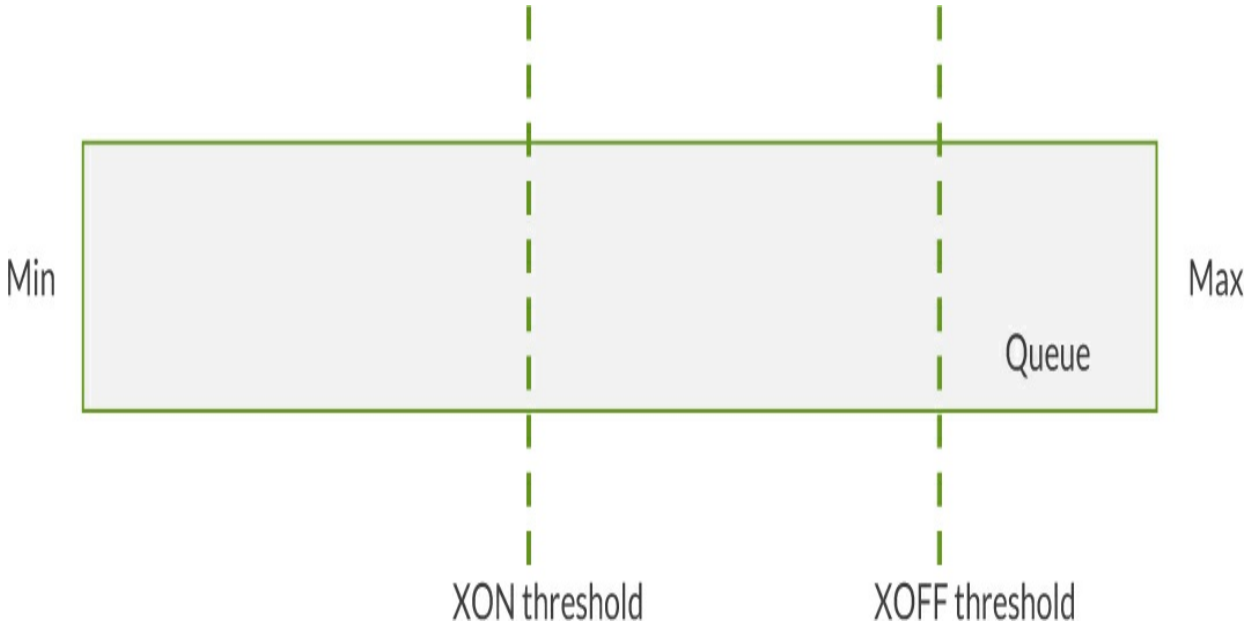


Figure 7-14 *XOFF and XON thresholds*

There are a few issues with PFC, though. The first problem is that PFC is designed to stop the traffic for the entire class experiencing congestion and not just the flow; this results in unfairness and head-of-line blocking, and it eventually leads to poor application performance.

Second, in some instances, the switch may receive excessive PFC pause frames from the downstream switch. The switch in turn propagates these PFC pause frames to its upstream neighbors toward the source. This results in a back-pressure mechanism propagating to the whole network, stopping the network traffic, and causing congestion and packet loss. This situation is referred to as a *PFC storm* and must be mitigated.

PFC watchdog is a technology that detects and mitigates PFC storms. PFC watchdog monitors the PFC pause frames received for each port and triggers mitigation actions when a PFC storm is detected. PFC watchdog has three functional blocks: detection, mitigation, and restoration. Figure 7-15 illustrates a simplified topology where PFC watchdog optimization is

implemented for an AI/ML data center fabric.

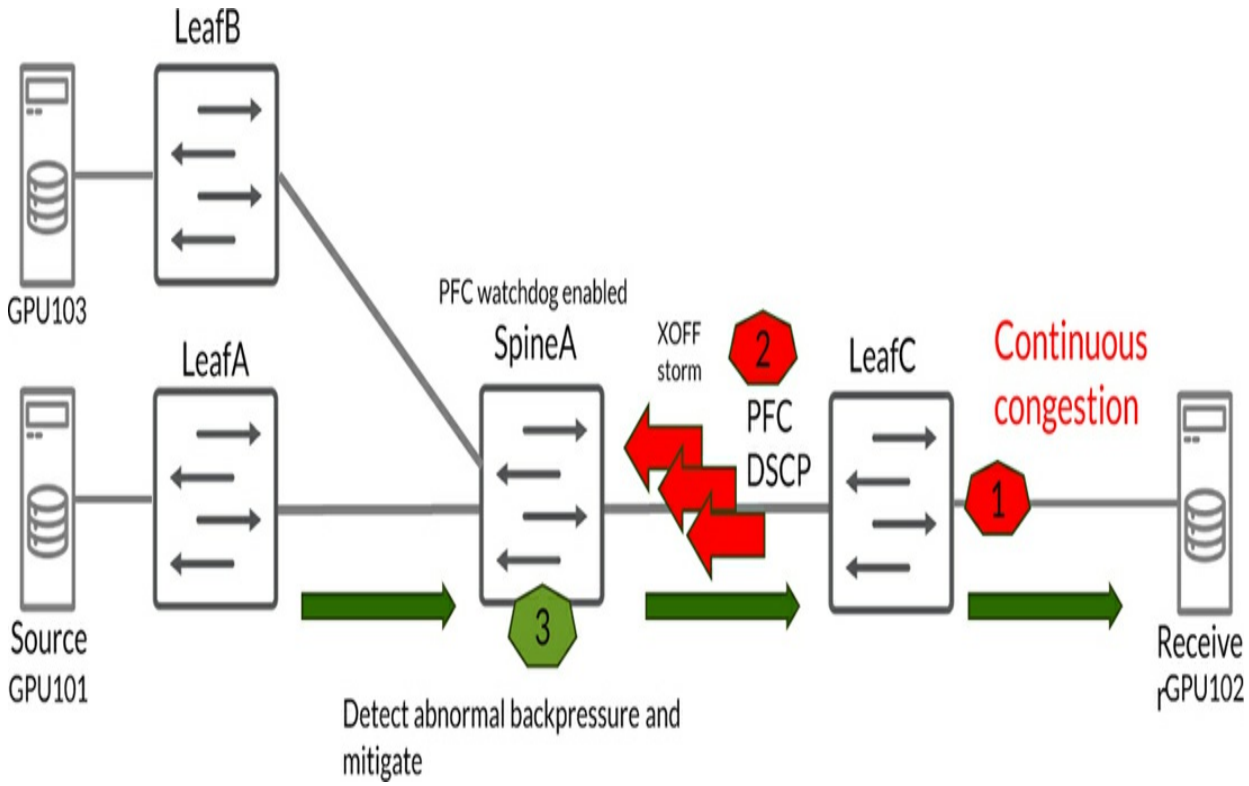


Figure 7-15 *PFC watchdog*

Figure 7-15 illustrates three steps:

1. **Detection:** SpineA receives PFC pause frames from LeafC, and as the queue continues to fill up, the pause frames get propagated. As LeafC continues to send PFC frames, SpineA continues to propagate them to all switches in the path toward the source (for example, LeafA). This results in a pause of traffic of the same class from LeafB. This, in turn, results in blockage conditions on all of the switches. PFC watchdog monitors the PFC pause frames received for each port and compares them with a configurable threshold. If the threshold is exceeded, the port is marked as being in a PFC storm.
2. **Mitigation:** When a queue has a PFC storm, PFC watchdog can choose to either drop or forward packets in that queue. If it chooses to drop, this is what it needs to do:

   • Discard all the packets that are already in the output queue.

- Discard any new packets that are meant for the output queue.
- Discard any new packets that come from the same priority group as this queue, including any pause frames. This limits propagation of the pause frame to the neighbor to signal congestion in this output queue.

Dropping is the most commonly used mitigation technique.

If PFC watchdog chooses to forward the packets, the queue ignores the PFC frames that are received. It forwards all the packets that are meant for the queue as well as the packets that were already in the queue.

Spine1 in Figure 7-15 receives PFC pause frames from LeafC, and as the queue continues to fill up, the pause frames are propagated. When PFC watchdog detects abnormality, Spine1 clears the queue and discards further PFC frames received from LeafC. As a result, the PFC pause frame is not sent to LeafA.

3. **Restoration:** PFC watchdog monitors the port that is impacted by the PFC storm and un-pauses the traffic when the received PFC pause frames are below a configurable threshold. This ensures that the device resumes lossless Ethernet functionality.

Data Center Quantized Congestion Notification (DCQCN)

As discussed earlier, both ECN and PFC face challenges with respect to performance. Whereas ECN is lossy, PFC is lossless, but it can result in unfairness. DCQCN combines ECN and PFC over DSCP to achieve better performance.

Figure 7-16 illustrates an example of ECN and PFC thresholds with DCQCN. ECN is normally configured below the PFC XOFF threshold. When the buffer utilization increases beyond the ECN threshold, the switch marks ECN bits in the packets of that flow. The packet continues its path to the destination. The destination acknowledges the ECN marking and generates a CNP toward the sender. The sender reduces the rate of traffic. If the buffer utilization continues to increase and crosses the PFC threshold, a PFC pause frame is triggered. The PFC frame is sent hop-by-hop upstream toward the source.

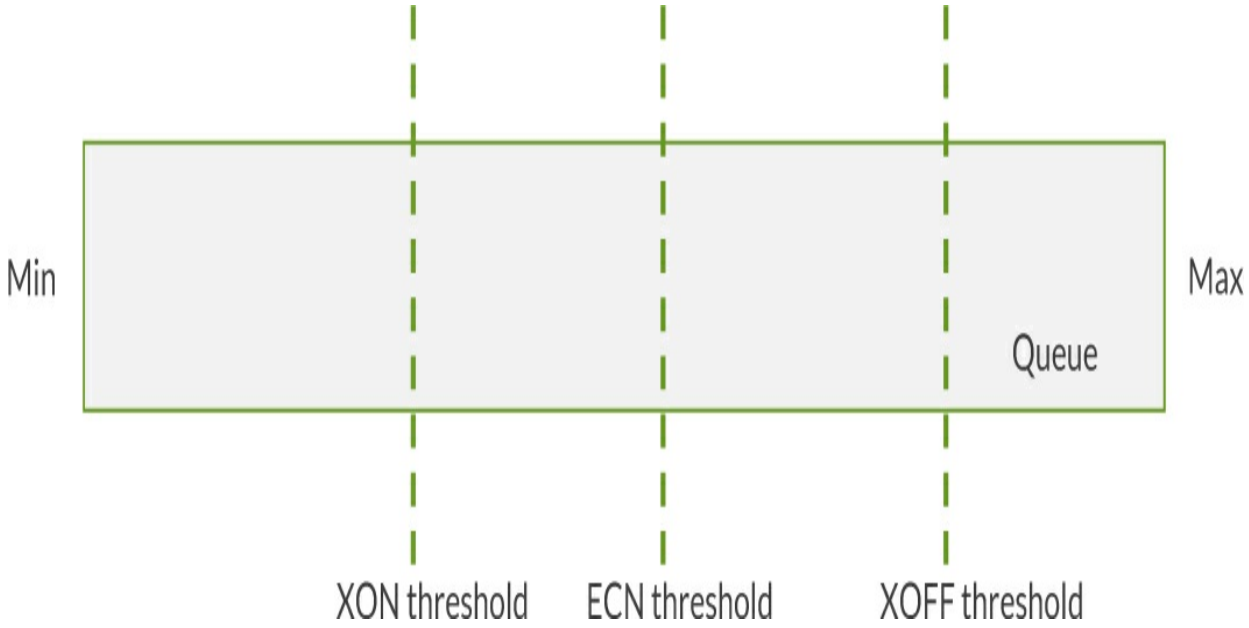


Figure 7-16 *ECN and PFC thresholds*

This is the sequence of steps that occur when the switch receives a PFC pause frame:

1. The PFC frame is received on an egress interface.
2. The switch pauses traffic in the corresponding egress queue.
3. A lossless buffer build-up causes a breach of the PFC XOFF threshold on corresponding ingress ports.
4. An ingress port generates a PFC pause frame toward the upstream node.

This continues hop-by-hop until the PFC reaches the sender, assuming that all transit nodes have PFC configured.

Figure 7-17 illustrates the DCQCN behavior. When the SpineA-to-LeafD queue buffer utilization exceeds the ECN threshold, SpineA marks the ECN bits in the packets and forwards it to the destination server attached to LeafD. As the server receives the ECN 11 bits, it acknowledges it with a CNP to the sender (the source of the write/read) attached to LeafA. Congestion mitigation occurs only when the sender receives the CNP and reduces the rate of traffic. In the meantime, if the buffer utilization crosses the PFC threshold, SpineA sends a PFC frame to LeafA that pauses the traffic on the egress

queue. LeafA pauses the flow of traffic for the class of traffic and sends a PFC pause frame to the source.

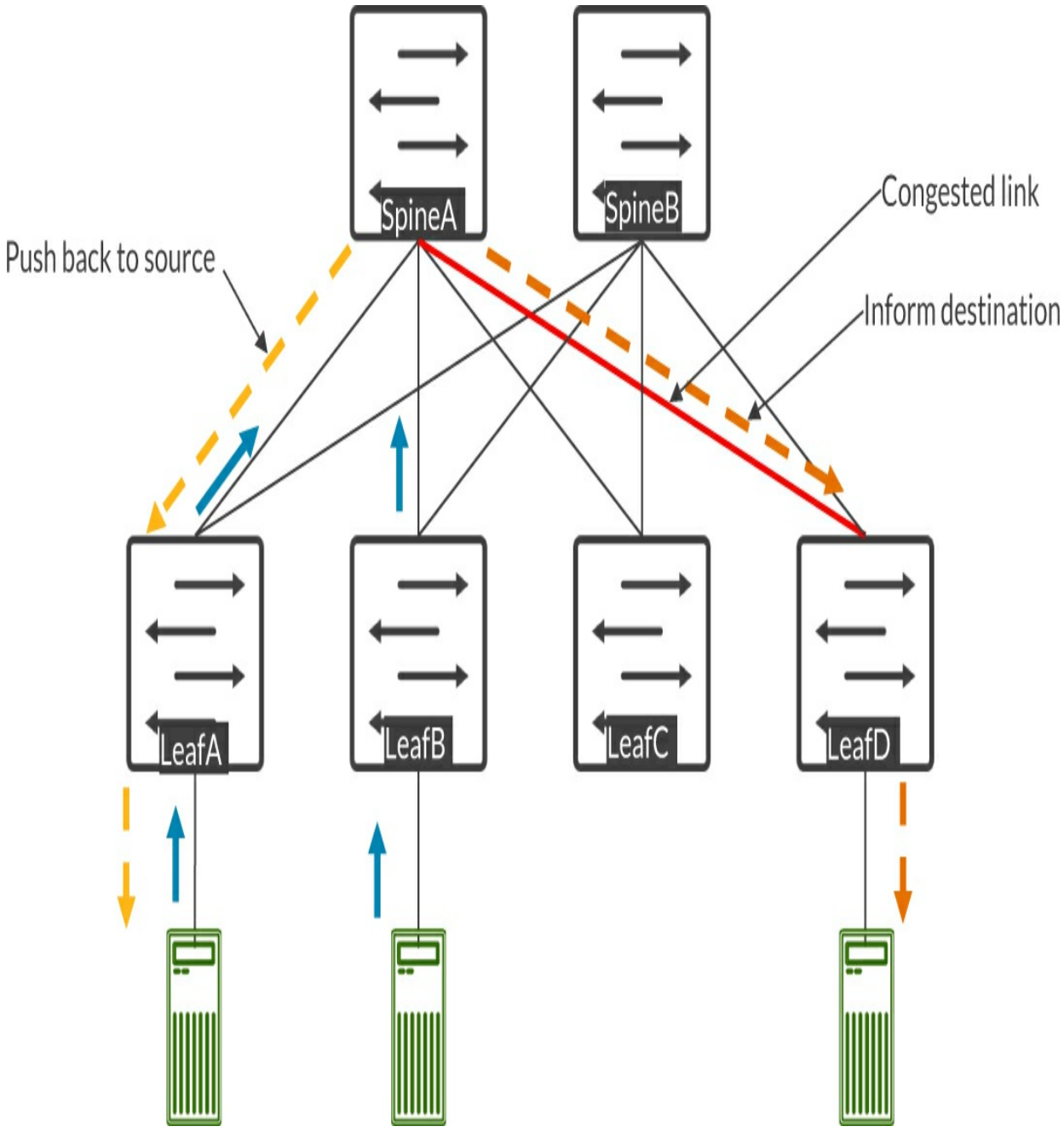


Figure 7-17 *DCQCN behavior*

In summary, while ECN congestion mitigation is slower than PFC, PFC results in a higher impact on traffic because it pauses the traffic for an entire class and not just the flow. Therefore, the ECN threshold is kept lower than

the PFC threshold. If the congestion situation is resolved by ECN, the other flows of the queue remain unaffected. But if the congestion situation worsens quickly, PFC is triggered, and it provides congestion mitigation before any packet loss occurs.

Most setups implement ECN for congestion control. The implementation for PFC varies. In some setups it might not be configured at all, while in others it may be configured only on leaf switches and servers. PFC may be configured on all nodes as well. The implementation depends on the environment and the service-level agreement (SLA) that is in place.

Source Flow Control (SFC)

Source Flow Control (SFC), which is specified in a new IETF standard (802.1qdw), is also referred to as Source PFC. Whereas PFC goes hop-by-hop, SFC directly sends the congestion signal from the congested switch to the source. The congested switch can trim the packet payload and reverse the source and destination in the header. The packet payload is then sent directly to the source. This prevents the congestion mitigation delay that occurs in ECN, where the ECN marked packet is first sent to the destination, which then sends a CNP to the source. Also, because SFC is flow based, it also prevents the class-level impact that occurs with PFC.

Figure 7-18 illustrates the flow of packets for SFC:

1. The server attached to LeafA initiates the packet flow.
2. LeafA forwards the flow toward SpineA.
3. Congestion is detected in the link between SpineA and LeafD, based on the queue usage and the SFC threshold.
4. SpineA sends SFC signals for the flow experiencing congestion. An SFC signal is created with the source and destination IP addresses reversed. This signal is destined to the source of the flow.
5. LeafA receives the SFC signal. It may intercept the signal and take actions such as pausing the traffic in the queue or take another action (for an area under exploration). LeafA can either forward the SFC signal downstream toward the server if the server NIC supports SFC or

can translate it to PFC before forwarding.

6. Upon receiving the SFC signal or PFC pause frame, the server pauses the flow for the duration marked by the timer field in the frame.

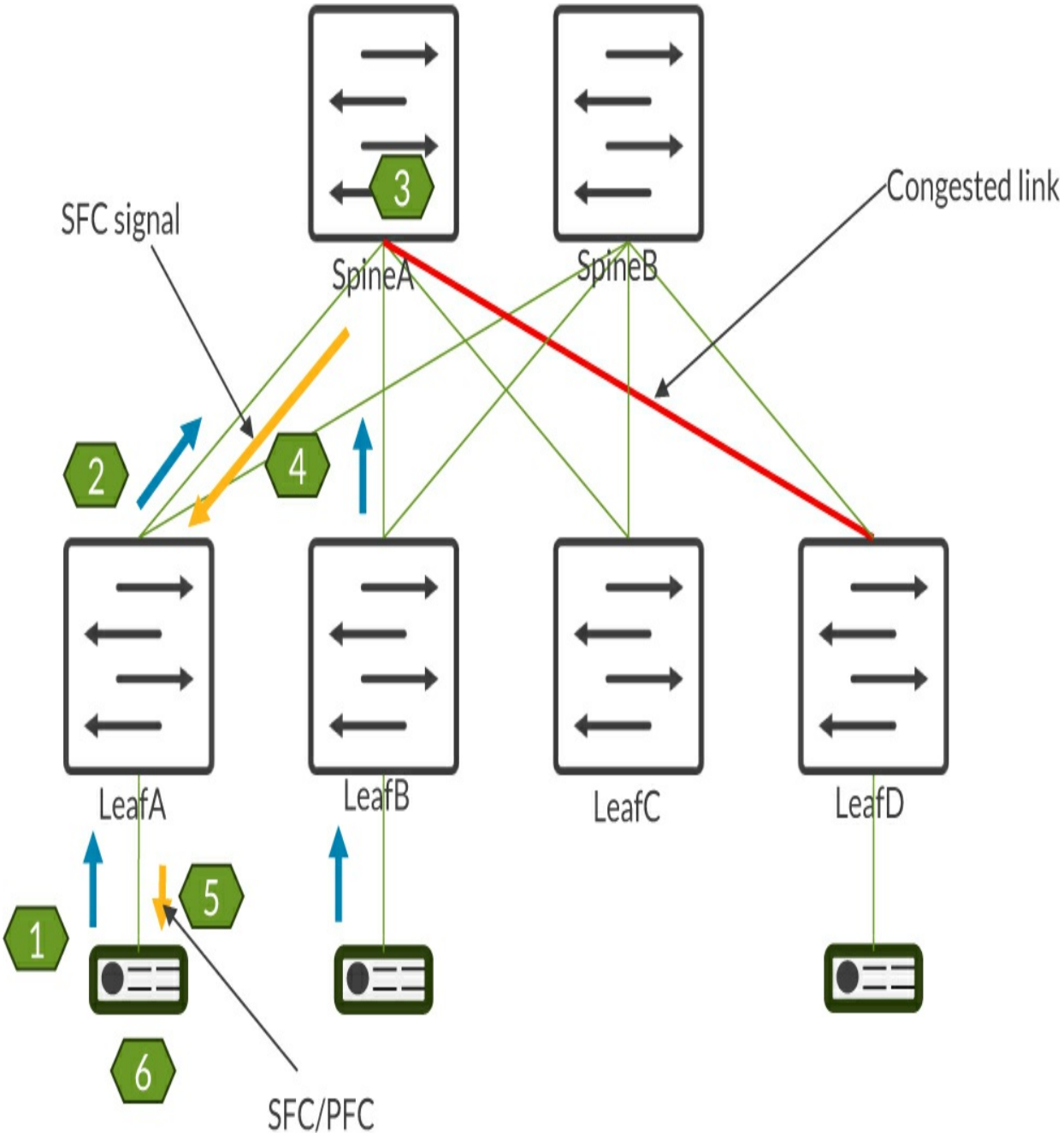


Figure 7-18 *Source Flow Control (SFC)*

Congestion Signaling

A new draft submitted to the IETF (draft-ravi-ippm-csig-01) talks about Congestion Signaling (CSIG). CSIG provides direct, real-time, in-band signals that network control loops can incorporate for performance and efficiency. It uses in-band network telemetry (INT) features on the switches that incorporate multi-bit signals in live data packets. It provides a simple low-overhead and extensible packet header mechanism to obtain fixed-length summaries from bottleneck devices along a packet path. This summarized information is collected over Layer 2 CSIG tags along the path. Receivers can reflect this information to senders via CSIG reflection headers. CSIG could be the next phase in congestion management that has traditionally been handled via PFC and ECN.

Figure 7-19 illustrates CSIG congestion mitigation. The CSIG signal is generated by the source server, GPU1. On each hop, the CSIG tags get added to the signal. Destination GPU2 receives the signal with CSIG tags from each hop. The destination server sends this signal with all the tags to the source. The source adjusts the traffic based on the information received.

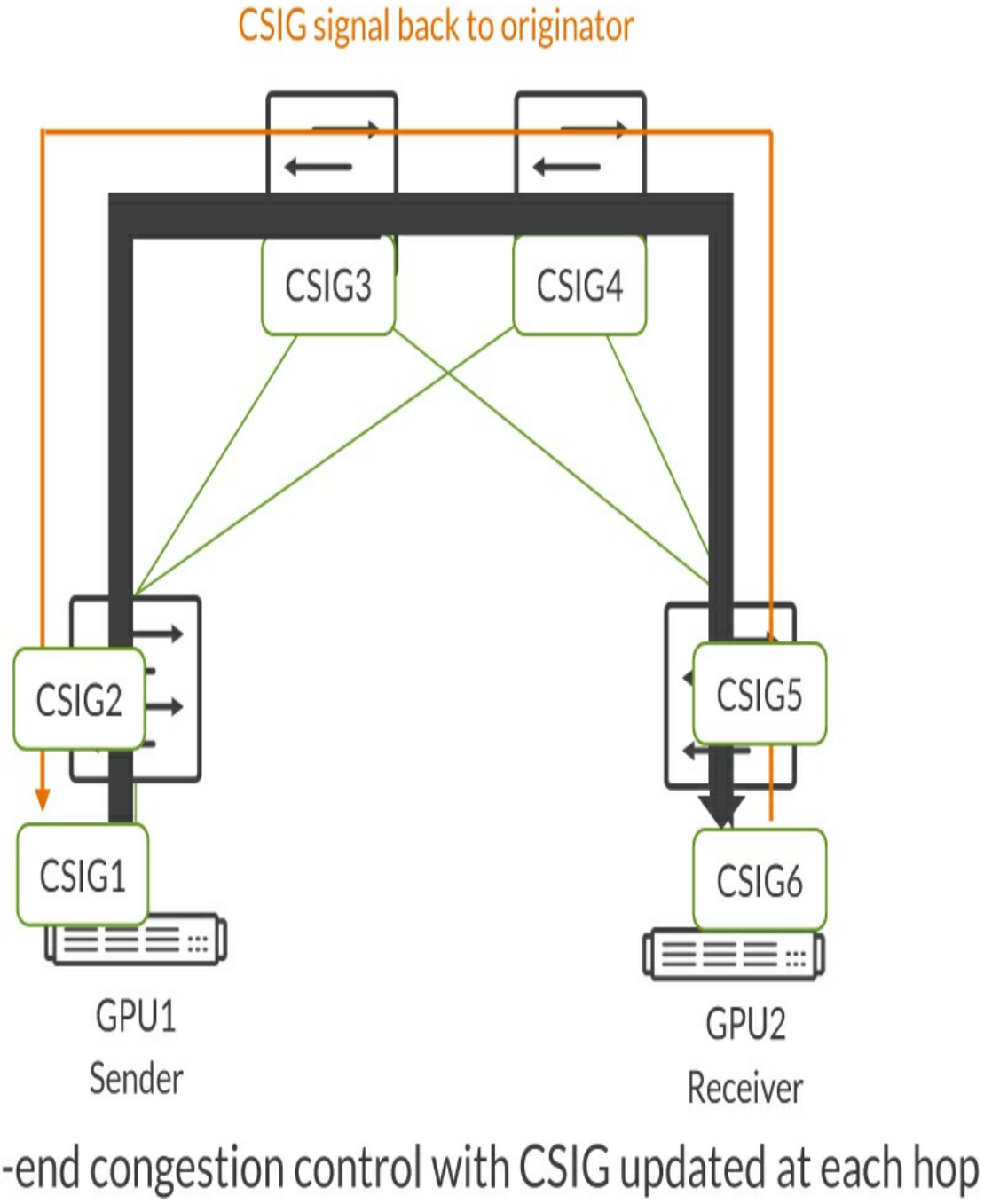


Figure 7-19 *Congestion Signaling (CSIG)*

These are some of the key points of the CSIG draft:

- A new CSIG tag includes congestion management information.
- CSIG quickly identifies path bottlenecks and suggests better path selection.

- A CSIG tag is between the Layer 2 and Layer 3 headers. Hence, the IEEE needs to allocate a new Ethernet tag.
- Servers are informed end-to-end via the CSIG reflection function.

Figure 7-20 illustrates the frame of a CSIG tag, ad Figure 7-21 shows the information contained in the CSIG tag.

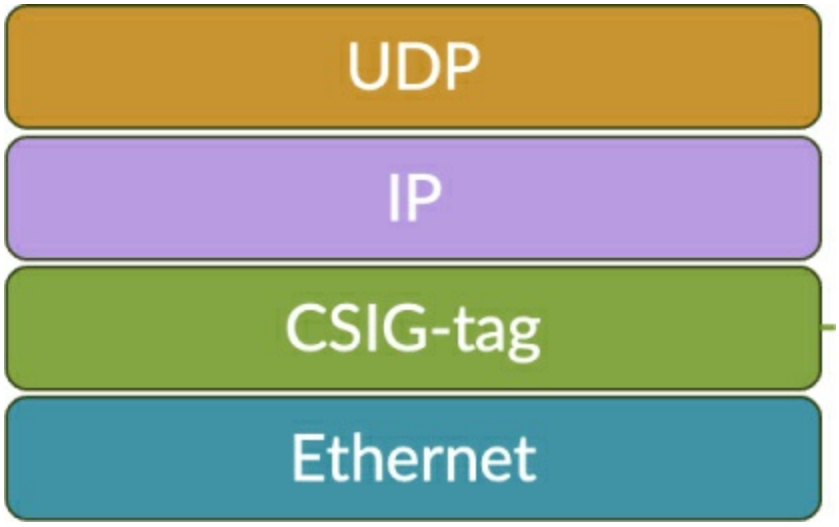


Figure 7-20 *CSIG frame*

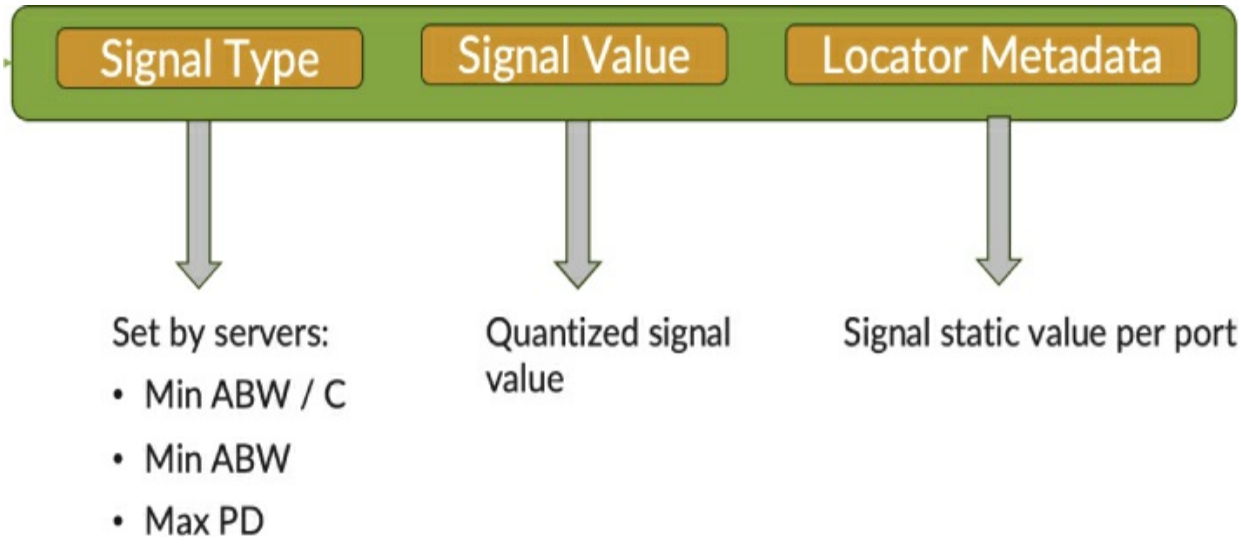


Figure 7-21 *CSIG tag information*

The locator metadata contains the following information:

- Capacity of the bottlenecks

- Stage of the bottlenecks
- Device ID
- Link identification (uplink or downlink)

Congestion Signaling is new but has good potential and will evolve with time.

Summary

In an AI/ML network, congestion can be mitigated using ECN (Explicit Congestion Notification), PFC (Priority Flow Control), or DCQCN (Data Center Quantized Congestion Notification) mechanisms. All these mechanisms avoid persistent congestion by sending signals to the source, which can then slow the data rate temporarily. The frequent use of the PFC over DSCP techniques may, however, result in a significant reduction in the speed of data exchange.

It is important to enable efficient load-balancing techniques that minimize the risk of congestion. Also, additional mitigation techniques such as the PFC watchdog must be implemented to avoid persistent deterioration of application performance that results from an avalanche of PFC pushbacks. Alternatively, in situations where PFC is likely to be triggered frequently, it is simply better to implement only ECN techniques and focus on efficient fabric load balancing, thereby reducing the probability of congestion.

Test Your Knowledge

Chapter Review

The following questions are designed to test your understanding of the content covered in Chapter 7. Following the questions, answers are provided so you can verify your conclusions.

Questions

1. What is RoCEv2, and how does it enable RDMA over Ethernet in AI/ML clusters?
2. Describe the main congestion points in RoCEv2-based AI data center networks and their impact on performance.
3. How does Explicit Congestion Notification (ECN) function in RoCEv2 fabrics, and what are its operational parameters?
4. Explain Priority Flow Control (PFC) and its limitations in lossless Ethernet fabrics for AI/ML.
5. What is Data Center Quantized Congestion Notification (DCQCN), and how does it combine ECN and PFC for end-to-end congestion control?
6. How does Source Flow Control (SFC) differ from traditional PFC, and what are its advantages in RoCEv2 fabrics?
7. Describe the role and operation of Congestion Signaling (CSIG) and in-band telemetry in RoCEv2-based AI fabrics.
8. Summarize the relationship between ECN, PFC, DCQCN, and the newer SFC in achieving lossless, high-performance RoCEv2 fabrics for AI/ML.

Answers

1. RoCEv2 (RDMA over Converged Ethernet version 2) encapsulates RDMA traffic in UDP/IP packets, enabling RDMA capabilities over standard Ethernet networks. It leverages lossless Ethernet features (for example, PFC, ECN) to provide low-latency, high-throughput, zero-copy data transfers between servers, bypassing the CPU and kernel. RoCEv2 is widely used in AI/ML clusters for distributed training, where rapid synchronization of large data volumes is required.
2. Congestion can occur at multiple points:
   - Local leaf link congestion: Multiple servers sending line-rate traffic to a local storage or compute node
   - Leaf-to-spine and spine-to-leaf congestion: Multiple flows

converging on the same uplink/downlink, especially in Clos topologies

- Leaf-to-server congestion: Oversubscription or bursty traffic exceeding NIC or server bandwidth
- Spine-to-super spine congestion: Aggregation points in multi-stage fabrics

Congestion at any point can cause packet loss, increased latency, and degraded JCT, especially for synchronized AI/ML workloads.

3. ECN enables end-to-end congestion signaling by marking packets when switch buffer utilization exceeds a configurable threshold. Marked packets trigger the receiver to send a congestion notification packet (CNP) to the sender, which then reduces its transmission rate. ECN thresholds must be carefully tuned to balance responsiveness and avoid excessive packet drops. ECN is effective for early congestion detection but may be too slow for microbursts or rapid congestion events.
4. PFC is a link-layer mechanism that pauses traffic for specific classes when buffer thresholds are exceeded, preventing packet loss. It operates on a per-priority basis, using XOFF/XON frames to control flow. Limitations include head-of-line blocking (pausing all flows in a class), risk of PFC storms (cascading pauses), and unfairness (one flow can block others). PFC must be carefully configured to avoid deadlocks and ensure fairness in high-throughput AI/ML environments.
5. DCQCN is a hybrid congestion control protocol for RoCEv2, combining ECN (for early, end-to-end signaling) and PFC (for lossless link-layer flow control). DCQCN uses ECN marks to adjust sender rates via a quantized feedback loop, reducing transmission rates before PFC is triggered. If congestion persists, PFC provides immediate lossless backpressure. DCQCN requires NIC and switch support, and its parameters (for example, ECN/PFC thresholds, rate reduction factors) must be tuned for optimal performance.
6. SFC (Source Flow Control) is a new IETF standard that allows congested switches to send congestion signals directly to the source, bypassing hop-by-hop PFC propagation. SFC can trim packet payloads

and reverse source/destination headers, enabling faster, flow-based congestion mitigation. Unlike PFC, which operates at the class level, SFC targets specific flows, reducing head-of-line blocking and improving fairness.

7. CSIG is an emerging mechanism that embeds multi-bit congestion signals in live data packets using in-band network telemetry (INT). Switches add CSIG tags indicating buffer occupancy, congestion stage, and device/link IDs. Receivers reflect this information to senders, enabling real-time, path-aware congestion management. CSIG supports fine-grained, low-overhead congestion detection and can inform adaptive routing and rate control algorithms.
8. ECN provides early, end-to-end congestion signaling, enabling proactive rate reduction. PFC ensures lossless transport at the link layer but can cause head-of-line blocking and PFC storms if overused. DCQCN integrates ECN and PFC, using quantized feedback to adjust sender rates and minimize PFC activation. SFC offers fast, flow-based congestion mitigation, reducing reliance on class-based PFC. Together, these mechanisms provide a layered approach to congestion management, balancing responsiveness, fairness, and lossless operation in high-throughput, synchronized AI/ML workloads.

Chapter 8. IP Routing for AI/ML Fabrics [This content is currently in development.]

This content is currently in development.

Chapter 9. Storage Network Design and Technologies [This content is currently in development.]

This content is currently in development.

Part 4: KPIs and Performance Monitoring

Chapter 10. AI Network Performance KPIs

Significance of Performance Benchmarking

Key performance indicators (KPIs) are very helpful for evaluating and moving technology in the right direction. To understand the importance of KPIs, let’s consider an example of the evolution of wheels. Wheels have evolved over time to meet the demand for transporting people and goods over long distances. Initially, humans walked and carried goods themselves. Then they began to use animals for traveling and carrying goods farther, but the amount and weight that could be transported were limited. The desire to transport more faster led to the invention of wheels, which were first made of stones. Pulling carts that had stone wheels required a lot of effort, and the speed was still speed. The desire for greater speed led to further evolution into wooden wheels, which were lighter, required less effort to pull, and could travel greater distance. But wooden wheels had a short lifespan and provided a rough ride, which led to the introduction of an outer band to absorb the shock and make the ride smoother. The band initially was made of leather, then metal, and then rubber. The outer band reduced the force needed to move, increased the distance that could be covered, and improved the overall ride quality. Since then, there have been many enhancements in the rubber as well as the rims, which have changed from wooden to metallic to alloys. Wheels continue to evolve.

When evolution happens, benchmarking can be helpful to measure the progress with each iteration. AI/ML workloads have become popular in recent years and have accelerated the growth. It is important to monitor the evolution of these workloads with metrics. Benchmarking provides a way to

compare systems based on standardized metrics. These metrics should cover different aspects, and the results should be consistent across runs. As evolution occurs, benchmarking helps to find the right system among various options being explored and designed.

The main metric for measuring the performance of AI data centers today is job completion time (JCT). In this chapter, we'll explore some different ideas and methods. With AI/ML data centers, it's not enough to just keep track of server availability and energy usage; there are few additional important KPIs that reflect the specific challenges these systems face. For example, AI data centers are facilities that house high-performance computing (HPC) systems and applications for AI and ML. These data centers need a lot of power, cooling, and network bandwidth to run the sophisticated and demanding workloads of AI and ML. Therefore, it is important to track and enhance the performance, efficiency, and scalability of AI data centers using suitable KPIs.

As we have discussed in previous chapters, an AI/ML data center consists of two main components—the model and the data—and there are KPIs tied to both of these components.

Performance Benchmarking for AI Data Centers

- **Model:** These are a few of the indicators of model performance:
- **Accuracy:** How well does the model perform its intended task? Common metrics include mean squared error (MSE) for regression tasks and classification accuracy for prediction tasks.
- **Precision and recall:** These metrics measure a model's ability to correctly identify true positives and avoid false positives and negatives. Precision is expressed in terms of the number of bytes, for example, as 8-byte, 16-byte, or 32-byte floating point numbers or integers.
- **Latency:** How long does it take for the model to generate a prediction or complete a task?
- **Data:** These are a few of the indicators of data performance:
- **Data efficiency:** What is the quality of the data used for learning? Data should have little redundancy and should cover as many scenarios as

possible. Variety helps improve the model and make it robust.

- **Training time:** This is the most important aspect of AI/ML training fabrics. Because data sets can be petabytes in size, processing can take a lot of time, and the goal is always to reduce that time. Thanks to new generations of GPUs and parallel processing, training time is being reduced.

For training workloads, time to train is the most important metric as it encapsulates how quickly it is possible to get to a trained model. High-precision training can result in achieving a solution faster. With lower precision, it may take multiple iterations to reach the right trained model, whereas with high precision, it might be possible to get there in one longer iteration. For inference, the metric could vary based on use case. OpenAI reportedly used 1,023 A100 GPUs to train ChatGPT, with a training time of around 34 days.

AI data center KPIs are metrics that quantify the performance, efficiency, and scalability of AI data center systems and applications. These are some of the common KPIs for AI data centers:

- **Throughput:** This KPI measures the number of operations or tasks completed per unit of time by an AI data center system or application. Throughput can be measured at different levels, such as node, rack, cluster, or data center. Throughput can also be expressed in terms of operations per second (OPS), floating-point operations per second (FLOPS), or queries per second (QPS).
- **Latency:** This KPI measures the time elapsed between the start and the end of an operation or a task for an AI data center system or application. Latency can be measured at different levels, such as node, rack, cluster, or data center. Latency can also be expressed in terms of milliseconds (ms), microseconds (us), or nanoseconds (ns).
- **Accuracy:** This KPI measures the degree of correctness or quality of the output or the result of an AI data center system or application. Accuracy can be measured using different metrics, such as precision, recall, F1-score, or mean average precision (MAP).
- **Power:** This KPI measures the amount of electrical energy consumed by an AI data center system or application. Power can be measured at

different levels, such as node, rack, cluster, or data center. Power can also be expressed in terms of watts (W), kilowatts (kW), or megawatts (MW).

- **Efficiency:** This KPI measures the ratio of the output or the result of an AI data center system or application to the input or the resource consumed. Efficiency can be measured using different metrics, such as throughput per watt (TPW), FLOPS per watt (FPW), or QPS per watt (QPW).
- **Scalability:** This KPI measures the ability of an AI data center system or application to maintain or improve its performance, efficiency, and accuracy as the input or the resource increases. Scalability can be measured using different metrics, such as speedup, efficiency, or strong scaling and weak scaling.

MLCommons for AI Data Centers

One of the difficulties of AI data center KPIs is the absence of standardization and comparability among different platforms, architectures, and applications. To overcome this difficulty, a group of industry leaders and researchers have established MLCommons, a nonprofit organization that aims to develop common and fair benchmarks for AI data center systems and software. MLCommons provides a set of tools and best practices for assessing and improving the performance of AI data centers.

MLCommons was established in 2020 by a group of experts and researchers in the field of AI and ML. The goal of MLCommons is to speed up the progress and deployment of AI and ML workloads by developing standard and equitable benchmarks, datasets, and best practices for AI data center systems and software.

MLCommons Initiatives

MLCommons has the following objectives:

- Allow fair comparison of different systems while still promoting ML innovation.

- Advance ML progress through fair and helpful measurement.
- Ensure reproducibility to guarantee reliable results.
- Benefit both the commercial and research communities.
- Keep benchmarking costs low so everyone can join.

MLCommons has three main initiatives:

- **MLPerf:** This is a suite of benchmarks that measure the performance of AI data center systems and software across different domains, such as computer vision, natural language processing, recommendation systems, reinforcement learning, and HPC. MLPerf provides standardized and rigorous methodologies, metrics, and data sets for measuring and comparing the throughput, latency, accuracy, power, and scalability of AI data center systems and software.
- **MLCube:** This is a set of tools and best practices that enable the portability and reproducibility of AI data center systems and software across different platforms, architectures, and environments. MLCube provides a common interface and a container-based framework for packaging, deploying, and running AI data center systems and software on any hardware and software configuration.
- **People + AI Research (PAIR):** This program supports the research and education of AI and ML by providing access to data, compute, and expertise for academic and nonprofit organizations. PAIR also fosters collaboration and communication in the AI and ML community by organizing events, workshops, and publications.

MLCommons Benchmarking Suites

MLCommons benchmarking suites aim to provide fair assessments of training and inference performance for hardware, software, and services. They all follow certain rules. To keep up with industry changes, MLPerf is updated regularly, new tests are periodically run, and new workloads are added to reflect the latest in AI.

MLCommons offers the following suites for benchmarking:

- **MLPerf Training:** This benchmarking suite evaluates how quickly systems in training models reach a desired quality metric. Different suites are created for AI/ML workloads and for HPC workloads.
- **MLPerf Inference:** This benchmarking suite evaluates how quickly systems can take inputs and generate outputs using a trained model. Different suites are created for data center, edge, mobile, and tiny workloads.
- **MLPerf Storage:** This benchmarking suite tests how quickly storage systems provide training data when a model is being trained.

A working group community of experts defines each benchmarking suite, and they set the fair benchmarks for AI systems. The working group specifies the AI model to use, the data set to run it on, the rules for modifying the model, and the metrics for measuring how fast the hardware runs the model. By using this AI model tripod, MLCommons AI systems benchmarks assess the speed of hardware and also the quality of training data and of the metrics of an AI model.

Benchmarking a Data Center for Machine Learning

Evaluating a data center's performance for machine learning involves using the MLPerf benchmarks and a standard set of tests from MLCommons that follow a structured multistep procedure. The objective is to provide a fair, open, and repeatable assessment of a system's capabilities on real-world AI workloads. The specific methodology varies between tasks like model training and model inference, but the general process is as follows.

1. **Benchmark selection and division:** The initial step involves choosing the appropriate benchmarking suite. For a data center, this will be either the MLPerf Training or MLPerf Inference data center suite. Within these suites, one of these divisions must also be chosen:
    - **Closed division:** This division is designed for direct comparisons of hardware and software. Participants are required to use the same model, data set, and official reference implementation. This is the standard path for vendors seeking to compare their systems directly.
    - **Open division:** This division is intended to showcase new research or

algorithmic advancements. It allows participants to use different models or training methods, as long as the system achieves the same predefined quality metric.

2. **System and software setup:** The hardware and software stack for the benchmark must be carefully configured. This includes:
   - **Hardware:** The hardware is the physical components of the system, such as servers, GPUs, CPUs, and the network interconnect.
   - **Software:** Software includes the operating system, device drivers, chosen machine learning frameworks (e.g., PyTorch, TensorFlow), and any proprietary optimization libraries.
   - **Reference code:** The official MLPerf benchmark source code for the specific workload must be acquired. This code, typically found on platforms like GitHub, includes the model architecture, data loading scripts, and a load generator utility for performance measurement.
3. **Benchmark execution:** The actual benchmark runs are automated by MLPerf tools.

   These tools are used for inference benchmarking:

   - **Load generation:** A utility known as LoadGen is used to dispatch inference requests to the system under test (SUT) in a controlled manner.
   - **Scenarios:** Data center inference is evaluated across two primary scenarios. The first scenario is offline, where all queries are delivered to the SUT at once to measure the maximum throughput (queries per second). The second scenario is server, where queries are streamed to the SUT following a Poisson distribution to evaluate latency and throughput under a stringent latency constraint.
   - **Metrics:** The key performance metric is throughput, measured in samples per second or queries per second. For the result to be valid, the SUT must also achieve a specified accuracy target. Both a speed test and a separate accuracy test must be conducted.

   These tools are used for training benchmarking:

   - **Time to train:** The core metric is the time required to train a model

to a predefined target quality metric (e.g., a specific accuracy level).

- **End-to-end process:** Timing begins with the data loading process and concludes when the model reaches the accuracy target. This comprehensive approach is used to evaluate the entire system, including the data pipeline, storage, compute power, and communication efficiency.
- **Distributed training:** Data center training benchmarks often involve the use of multiple servers and accelerators to train models at scale and assess the system's ability to scale efficiently.

4. **Results and code submission:** Once the benchmark runs are completed and validated, the results, along with the detailed system configuration and all the source code used, are submitted to MLCommons. The submitted code must be made available to the public.
5. **Peer review and publication:** The final, critical step for ensuring fairness is the review and auditing of the submissions by MLCommons and its member organizations. The goals of this peer-review process are to verify that the rules were followed, the accuracy targets were met, and the results are reproducible. Verified results are then published in a publicly accessible, interactive table on the MLCommons website, https://mlcommons.org.

Summary

AI data centers are crucial for the development and innovation of AI and ML. However, they also present significant challenges and opportunities for assessing and improving their performance, efficiency, and scalability. AI data center KPIs and MLCommons are two key elements that can help tackle these challenges and opportunities by offering common and fair measures, data sets, tools, and best practices for AI data center systems and software. By using AI data center KPIs and MLCommons, AI data center operators and users can enhance their knowledge, testing, and comparison of AI data center systems and software and, ultimately, improve their AI and ML abilities and results.

Test Your Knowledge

The following questions are designed to test your understanding of the content covered in Chapter 10. Following the questions, answers are provided so you can verify your conclusions.

Questions

1. What are the most critical KPIs for evaluating AI data center performance, and how are they measured?
2. How does MLCommons's MLPerf benchmark suite standardize AI performance evaluation?
3. Why is job completion time (JCT) a key metric for AI/ML clusters, and what factors influence it?
4. How do throughput and latency metrics differ in their significance for training versus inference workloads?
5. What is the role of power- and energy-efficiency metrics (for example, FLOPS/Watt) in AI data center benchmarking?
6. How does scalability (strong and weak scaling) affect AI data center design and KPI interpretation?
7. Why is reproducibility essential in AI benchmarking, and how is it enforced in MLPerf?
8. How can benchmarking results inform architectural and operational improvements in AI data centers?

Answers

1. Key KPIs include job completion time (JCT), throughput (FLOPS, IOPS), latency (per operation and tail), accuracy (model quality), power consumption (Watts, PUE), and scalability (speedup, efficiency). These are measured using standardized benchmarks (for example, MLPerf), telemetry data, and application-level metrics. Accurate measurement requires synchronized clocks, representative

workloads, and consistent test environments.

2. MLPerf provides closed-division (fixed model/data set) and open-division (innovation allowed) benchmarks for training, inference, and storage. It specifies reference models, data sets, accuracy targets, and test harnesses, enabling fair, reproducible comparisons across hardware, software, and cloud services. Results are peer reviewed and published for transparency.
3. JCT measures the total time to train a model to a target accuracy. It is influenced by compute performance, network bandwidth and latency, storage throughput, parallelism efficiency, and tail latency. Reducing JCT accelerates model iteration and deployment, directly impacting business agility and research productivity.
4. Throughput (samples/sec, FLOPS) is critical for training, where large batches are processed in parallel. Latency (response time) is more important for inference, especially in real-time applications. Both metrics must be balanced to optimize resource utilization and user experience.
5. Power efficiency determines operational cost and sustainability. Metrics like FLOPS/Watt and PUE (power usage effectiveness) quantify how effectively hardware converts electrical power into useful computation. Energy-efficient designs reduce cooling requirements and environmental impact.
6. Strong scaling measures how performance improves as more resources are added for a fixed problem size; weak scaling measures performance as both resources and problem size increase proportionally. Poor scaling indicates bottlenecks in network, storage, or software, guiding optimization efforts.
7. Reproducibility ensures that results are reliable and comparable across systems and over time. MLPerf enforces this by requiring open-source code, fixed data sets, accuracy validation, and peer review. This builds trust in published results and drives industry progress.
8. Benchmarking identifies bottlenecks (for example, network congestion, storage latency), validates design choices, and guides investment in hardware/software upgrades. Continuous benchmarking enables data-

driven decision-making for scaling, tuning, and evolving AI infrastructure.

References

Chuan Li, “OpenAI’s GPT-3 Language Model: A Technical Overview,” June 3, 2020, https://lambda.ai/blog/demystifying-gpt-3.

Nvidia, “MLPerf Benchmarks,” https://www.nvidia.com/en-us/data-center/resources/mlperf-benchmarks/.

MLCommons, “Benchmarks” https://mlcommons.org/working-groups/

Chapter 11. Monitoring and Telemetry

Exploring Monitoring Options

Monitoring a data center's health is critical and is not just limited to the network infrastructure. Power, temperature, and humidity monitoring are crucial for the efficient operation of the data center infrastructure and involve many data points taken from sensors and other hardware devices. In this chapter, we focus on IP networking infrastructure monitoring and telemetry, which may sometimes include power consumption statistics but is mainly limited to networking infrastructure monitoring that is specific to an AI data center.

Different types of monitoring are commonly used in AI data center networks. The networking equipment supports Syslog, SNMP and telemetry for exporting the data from the devices. The same mechanisms are being used for the AI data centers. The data export mechanisms remain the same and are evolving more towards telemetry. The advent of AI data centers presents a challenge of exporting data faster so that decisions are taken and applied to the devices at the correct time. Exporting the data at the interval of seconds or milliseconds is not good enough, it has to go to micro and nano seconds intervals. Even the SNMP trap generation could be faster to handle the scenario of microbursts.

Let's look at some of the most common AI data center monitoring tools, which are sometimes used in parallel on networking devices such as switches, routers, and firewalls in the backend AI data center as well as in the frontend network.

- **SNMP polling:** With SNMP polling, SNMPv2c or the more secure SNMPv3 is used to look every couple seconds or minutes at a specific

MIB. SNMP polling can focus on interface stats, queue stats to get the total current counter or the current rate per second, the total buffer utilization at the switch of the shared or dedicated buffers, per-interface buffers, and queue buffers. It can also focus on node capacity information for the FIB, RIB, or MAC table; TCAM utilization; or standard performance of the node at the CPU level, the memory used globally at the node, or the CPU and memory of the individual line card and ASIC of the switch.

- **SNMP traps:** SNMP traps are generated based on the threshold set for specific parameters on the device. For example, an SNMP trap may be sent to the collecting node when the CPU of the device reaches a certain level of utilization. Or, in lossless fabric, an SNMP trap may be generated by a device automatically when the rate of the Priority Flow Control (PFC) pushbacks received on the interface on the specific queue is reached.
- **Syslog monitoring:** Syslog monitoring may be carried out for specific Syslog messages that are reporting critical or fatal errors for global device parameters or for specific interface-level problems.
- **Telemetry streaming interface:** Instead of just using the legacy SNMP-based polling approach, a device might send information on specific statistics to a telemetry collector— either periodically or when a change occurs. Telemetry uses a push mechanism instead of a pull method (SNMP). Telemetry data can be pushed by the forwarding engine directly as well via the revenue ports. Periodic exports require continuous monitoring of the devices from the collectors and data has to be sent from the device continuously. Change based telemetry is a method where the networking device exports information only when there is a change from previous state or value. For example, if there’s no change in a statistic, why would the telemetry interface continue to send the same information to the collecting device? In some cases, on-change telemetry is more suitable. Telemetry streaming may provide global node-level information, or it may operate per-interface or per-queue. Global telemetry streaming may export information related to buffering, shared versus dedicated monitoring, monitoring of the CPU, node memory, the Forwarding Information Base (FIB), the Routing Information Base (RIB), or TCAM utilization (which is related to the

use of access lists to protect the IP network from various attacks). Interface-level telemetry provides details on buffer utilization (with accuracy at the per-queue level), the rate of packets per interface, and cumulative statistics for IPv4 and IPv6. The telemetry interface uses the gRPC or gNMI approach, allowing for the simultaneous establishment of multiple concurrent sessions from a given device to different collectors.

- **In-band flow analyzer:** An in-band flow analyzer can be used with on-data plane metadata information or data plane cloning probes to measure latency and congestion in the data center fabric.
- **RPC query:** Like SNMP polling, an RPC query can help automate information gathering at the command-line interface with XML, JSON or similar structure.
- **Agent-based network monitoring:** A dedicated agent runs TWAMP probes to measure RFC 2455 benchmarking of the network. Various probes can be used, including UDP- or TCP-specific probes, and latency and loss can be measured.
- **sFlow and IPFIX:** These tools are used to monitor flow characteristics and analyze applications used over the network.

Both historical and real-time monitoring are used to analyze data center networks.

With historical network monitoring, the statistics data trend is identified, and corrective actions are taken on the network isolated node or specific interface. The collected historical data helps predict the situation. For example, it might be possible to see that bandwidth utilization grew 10% over the course of a couple weeks, and we might believe there’s a good chance it will continue to grow in the next couple weeks, assuming that applications continue to be deployed in the new data center or more customers use the infrastructure.

The large language model (LLM) is used to analyze the historical data, and the generative AI takes corrective actions even before the problem appears; the AI agent suggests taking corrective action based on the monitored data.

Real-time network monitoring is typically streaming telemetry based instead of polling based. Data is collected from the data center Ethernet/IP switches,

routers, or firewalls to get a better situational awareness. Real-time network monitoring uses real-time data statistics for flow tail latency; these statistics include packet drops per queue, fat flow information, and entropy scores for the flow. Real-time network monitoring seamlessly moves part of the server traffic to other nodes of the Ethernet fabric.

Real-time network monitoring involves analyzing real-time data for a given network node; the data might include memory utilization of the ASIC, CPU of the switch/router/firewall nodes, or remaining storage space. Corrective action might involve reducing the load on a node by blocking specific sources if the origin of the churn was related to a security breach. If the spread of an incident could evolve, it is important to have a plan to evacuate the traffic to a different redundant network switch or router or a different data center site and to take the corrective actions without any live traffic.

Network Monitoring in an AI/ML Data Center Network

InfiniBand technology has been the most used technology when it comes to lossless fabric. Ethernet has undergone significant improvements in telemetry and monitoring over recent years and is even more advanced in Ethernet AI data center scenarios compared to the traditional InfiniBand networks. In an AI data center, RDMA is considered as a technology that can be enhanced to achieve similar performance with ethernet. The monitoring and telemetry allow you to determine if lossless Ethernet fabric delivers on its promises after deployment and provides the similar performance experience as InfiniBand with Ethernet fabric.

The collector software that is involved in network monitoring is necessary to handle real-time data from switches in telemetry streaming mode. In the AI/ML data center context, however, it’s more helpful to have a switch produce telemetry data as well as the server telemetry, to allow greater correlation of the server and switch telemetry data to identify the performance consistency or the changes trends across the entire system.

Advanced software that examines and connects different indicators is crucial to the success of AI data center monitoring. With many data collection tools, having a large amount of data is beneficial. However, the primary goal is to

ensure that we understand how to utilize the information collected from various network nodes and correlate the data across different points within the network at different points in time. For example, within data center fabric, frame loss or latency differs at the aggregation point compared to non aggregation point, such as a spine, compared to a leaf, which connects only a subset of servers.

With telemetry data, it is possible to collect vastly more data than is possible with the traditional SNMP polling mechanism. In addition, use of telemetry data requires more ability to store information on servers, and servers need to be able to analyze real-time data. Also, more resources are required on the server side, including CPU memory to process data in real time and create visualizations such as graphs, also in real time. The metadata in the telemetry data received by the collector server is also typically more significant and more accessible for comparison between nodes.

In the context of AI data centers, monitoring of network utilization focuses on the following aspects:

- **Utilization of egress buffers:** Monitoring the utilization of egress shared buffers or dedicated egress buffers is essential even if applications are not yet experiencing related issues. This monitoring can enable you to take corrective action such as adding more bandwidth between leaf and spine or moving part of the server traffic to other links (assuming that you can run some form of traffic engineering inside the fabric).
- **Latency changes:** In-band telemetry (with probes and timestamps) makes it possible to observe latency changes between different leaf nodes in the network and ensure consistency in latency across all leaves in the fabric.
- **Bandwidth utilization:** By monitoring bandwidth utilization trends from leaf to spine and spine to leaf, you are equipped to determine whether any spine is getting more overutilized. Overutilization can be a symptom of load balancing issues related to an increasing number of elephant flows in the network with lower flow entropy.
- **Application flows:** It is possible to analyze application flows by using sFlow information.

- **Mirroring on demand:** Mirroring packets on demand allows you to analyze user content and examine dropped packets.
- **Mirroring on the drop:** Mirroring on the drop is also helpful to make sure you know what kind of packets are getting dropped or were dropped. You can collect dropped packets at the collecting station and use them to determine if drops were related to application malformations or the security rules in place. The packet drops in a network are not all bad; sometimes they occur because the network is defending itself against an attack or because the control plane or data plane load on the network is too high, and the network starts to protect itself. Perhaps DDoS is triggered at the switch ASIC because too many ARP requests are observed per second. In response, the switch may begin dropping packets randomly, or egress queue loss may occur on lower-priority queues.

Even with numerous monitoring options available, it may still be impossible to react to a performance degradation trend quickly. For example, when network congestion happens at the microsecond level, a monitoring station may not have a sufficient level of accuracy to capture and report it correctly.

On the other hand, even if the occurrence and reporting of some negative symptoms can’t be fixed immediately (with automated corrective actions), at least the origin and size of the problem will be known, allowing the network administrator to make an informed decision. Is the network constantly dropping packets? Is it happening on a single interface or on many interfaces? Is the problem isolated to one switch node, or is it occurring on multiple switches simultaneously?

It is important to know whether you need to be able to react immediately with corrective actions or whether the server application is robust enough to handle drops. In the context of an AI data center, for example, monitoring might report the following types of real-time statistics:

- **Frame loss:** You might get reports about the egress per queue frame loss reported via telemetry or SNMP.
- **PFC stats**: You might monitor the PFC XOFF backpressure augmented rate across all the queues and on many interfaces of the spines or leaf nodes.

- **Latency:** The latency of the sample traffic may be higher than usual.
- **ECN stats**: Explicit Congestion Notification (ECN) stats may be increasing on all queues, and many GPU servers may be slowing down their learning process and reporting higher JCT numbers than usual.

Depending on the magnitude of a problem reported in real time, the following worst-case situation decisions may need to be made:

- You may need to shut down the server at the origin of the congestion (in which case it's helpful to have switch software that is capable of identifying the origin)
- You may need to restart the learning process at the scheduler level.
- You may need to evacuate the entire data center to a location that has better network capacity and restart the jobs in that new location.

Monitoring the network and taking minor corrective actions won't help if overall network capacity planning was not done correctly—for example, if the oversubscription ratios were incorrectly designed, the wrong packet load-balancing method was implemented on the switches, or the quality of the optics was not suitable for the 800 GbE or 400 GbE interface connection. You need to ensure that monitoring is done on a AI data center network that was correctly designed and included the proper capacity planning analysis to support the target GPU connection and the target number of concurrent jobs in the backend network.

In-Band Flow Analyzer (IFA)

Earlier we mentioned monitoring GPU workloads and the types of corrective actions that can be taken. What if you could add extra metadata to a data packet to gain better visibility into performance, or what if you could add specific probes to the network to measure it with greater accuracy? This is precisely where IFA-based telemetry comes into play.

Besides doing traditional real-time monitoring, you can include more information in the original data packets or add specific probe packets (which are copies of the original packet) to the data plane network to better measure its performance (such as its latency). If you use IFA and set the measurement

metadata on the original packet or a copy of the original packet as a probe, you have a good chance of getting a more accurate latency measurement than if you used an artificial agent, which has a different packet payload content compared to the original user flow. With IFA, the original data plane packet is tagged with info, or a packet clone is used for probing purposes.

So, IFA enables you to tag packets in a flow with metadata, and then the server can analyze this information to detect any network faults and bottlenecks. The IFA header on a packet is set to tag the packet accordingly at each hop along a network. The initiator, transit, and termination nodes are well identified with IFA. A data center fabric typically uses a three-stage IFA model or a five-stage IFA model (for rail-to-rail communication).

IFA consists of the following nodes:

- **Initiator:** The initiator IFA node samples the traffic of interest and converts it to an IFA-capable flow with specific header information. Sometimes, depending on the implementation, specific traffic may be cloned and set with IFA metadata dedicated to the measurements in real time; for example, the initiator node might add timestamps and node information to every cloned packet traversing the fabric for the given flow. The node also adds the initiator tag so that the collecting server correlates all the information received in the metadata, from the number of hops to the timing information.
- **Transit:** The transit IFA node, which may be a spine or a super spine in the data center fabric, appends metadata with local information. It adds local updates to the metadata stack information.
- **Termination:** The termination IFA node is typically an egress leaf node or a border-leaf node. When the IFA metadata is not set on cloned traffic, the metadata information is removed; when it is set on cloned probe traffic, the metadata stack of information is sent to the collecting IFA server application (for example, in the IPFIX format), but the cloned data packet is typically dropped at the egress switch in the IFA telemetry system

Figure 11-1 illustrates the IFA process.

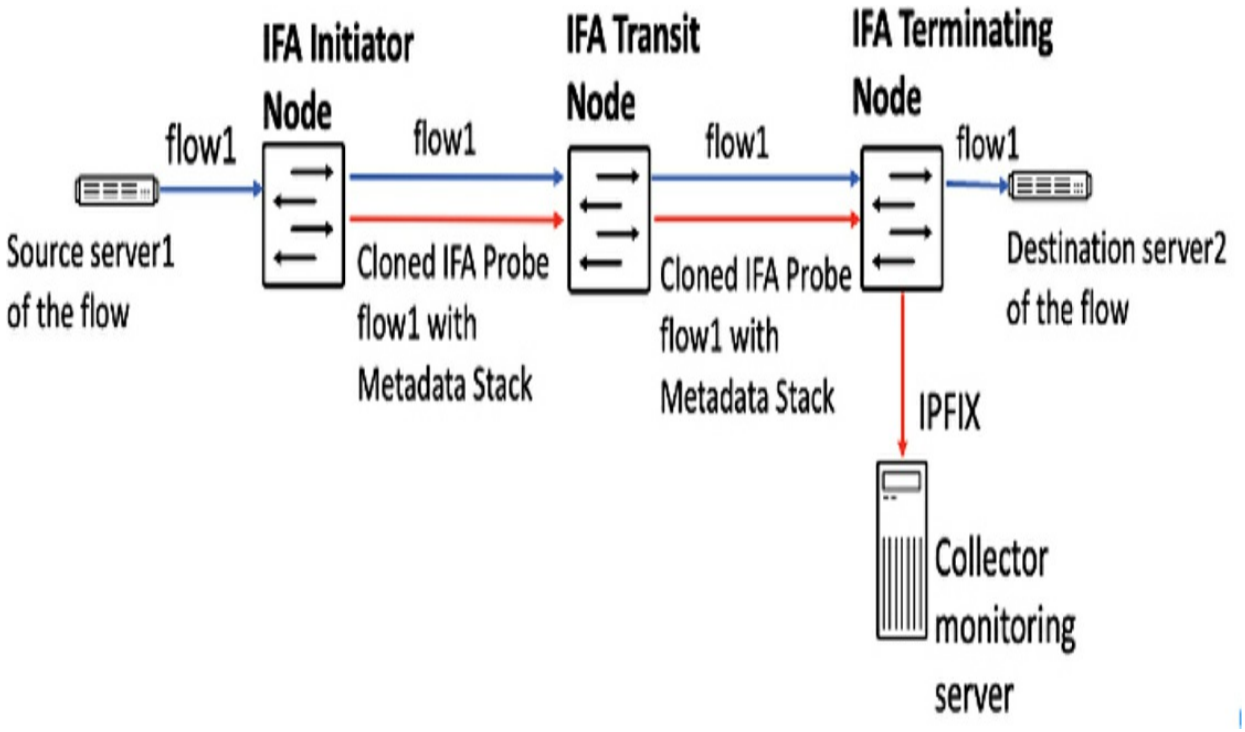


Figure 11-1 *IFA for in-band telemetry monitoring*

IFA collects data such as the following:

- Residence time (how long a packet spends within a single hop)
- Per-hop latency (cumulative latency for a packet to travel between two adjacent network devices which includes residence time plus the time on wire)
- Per-hop ingress port number
- Per-hop egress port number
- Received packet timestamp value
- Queue ID
- Congestion notification
- Egress port speed

Figure 11-2 illustrates the L3 data packet format with the IFA metadata to highlight that new portions are stacked before the original payload appears inside the packet.

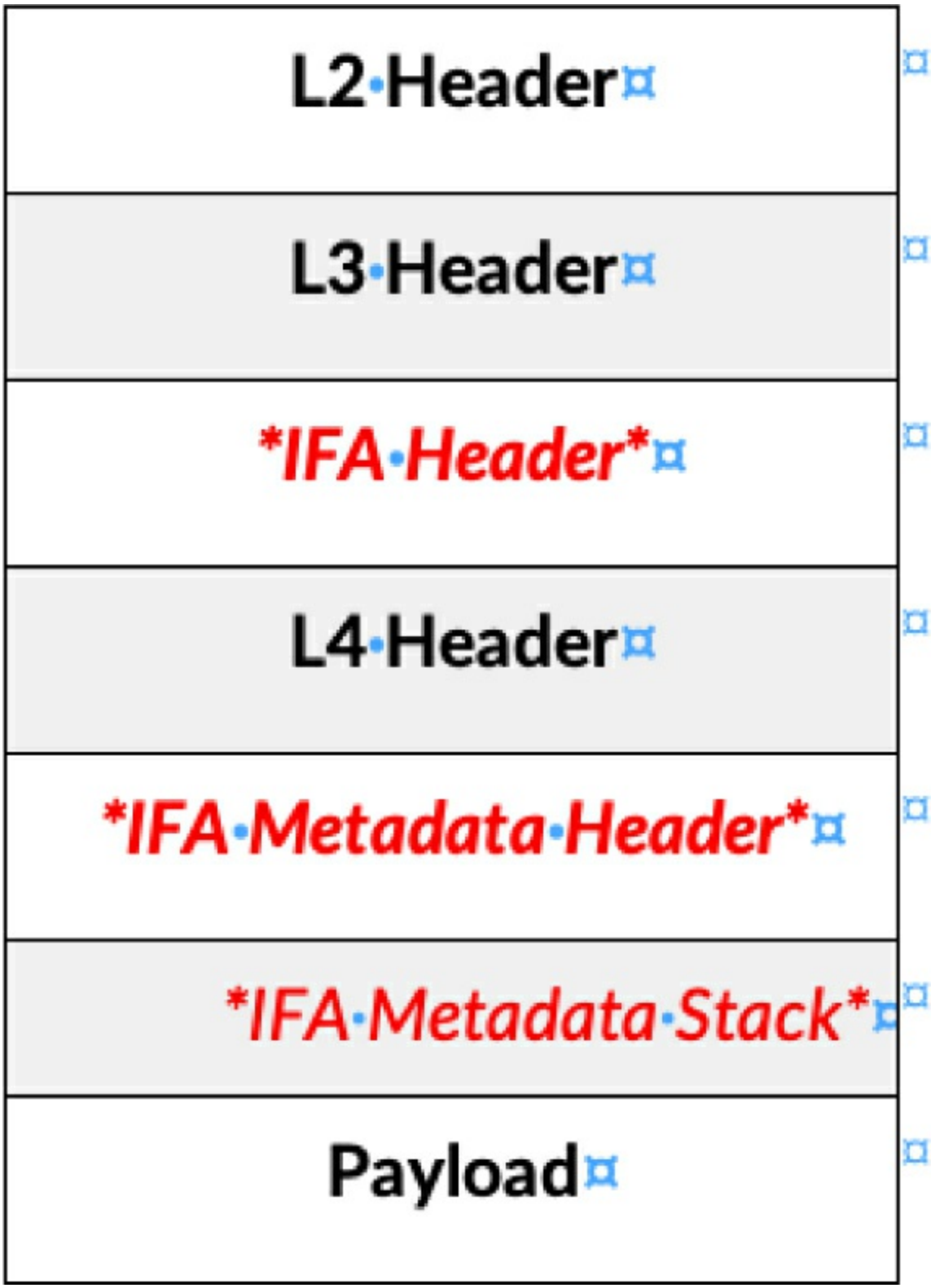


Figure 11-2 *IFA packet format with metadata information*

As you can see in Figure 11-2, the IFA header is the first header after the IP

header. The MTU of the interfaces in the fabric must be large enough to support additional header of IFA. This is typically not a big problem, and most switches in a data center have the MTU size set to 9000 bytes or more.

The IFA header identifies the packet. Figure 11-3 shows the contents of the IFA header.

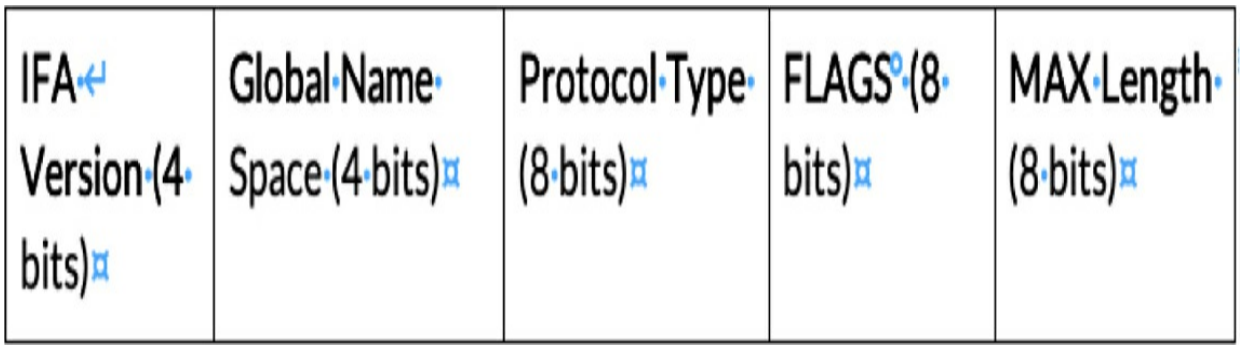


Figure 11-3 *IFA header format*

Just after the IFA header is the original L4 transport header, but then comes the IFA metadata header. This header is pushed by the initialization node. Metadata header contains information on the action to be performed on the given IFA packet: either by transit or termination egress node. It also includes the hop limit, in case the initiator would like to reduce the scope of the IFA packet at the initiator itself. Then the most important part of the IFA packet is the IFA metadata stack, which contains relevant information.

Figure 11-4 shows the contents of the metadata stack.

Local Name Space (4 bits)	Device ID (20 bits)		IP TTL (8 bits)
Egress port speed (4 bits)	Congestion (2 bits)	Queue ID (6 bits)	Rx Timestamp Seconds (20 bits)
Egress Port Number (16 bits)			Ingress Port Number (16 bits)
RX Timestamp Nano Seconds (32 bits)			
Residence Time Nano Seconds (32 bits)			
Opaque data 1 (Reserved) (32 bits)			
Opaque data 2 Hi (Reserved) (16 bits)			Opaque data 2 Low (Reserved) (16 bits)
Opaque data 3 (Reserved) (32 bits)			

Figure 11-4 *IFA metadata stack contents*

For a complete description of each field, you can see the IETF document https://www.ietf.org/archive/id/draft-kumar-ippm-ifa-08.txt. For now we will focus on fields related to receipt timestamps, nanoseconds, congestion, and residency time.

For an AI data center, it is important to know whether a packet experienced congestion; in order to know this, though, ECN must be enabled on the egress port. The residence time, in nanoseconds per hop, is a latency value that depends on a given switch node's capacity to calculate it; therefore, it is sometimes best to keep nodes with the same computational capacity at the given architecture level in order to make the latency stamp accurate. Latency is usually calculated based on the metadata received, by subtracting the receipt time from the transmit time.

The IFA stack also contains other information, such as device ID, port speeds, and port numbers. However, depending on the implementation, they may not be used, and only the latency and congestion information may be relevant.

To summarize our discussion about IFA in the context of an AI data center, these are the main benefits:

- IFA probe packets traverse the same network path as the original flow, helping you monitor the network for faults and performance issues.
- IFA monitors live traffic and thus helps to perform packet-level latency analysis and queue-congestion monitoring to optimize network performance.

As with all other technologies, there are limitations with IFA that will get addressed as the feature matures. For example, for general networking purposes, the way multicast traffic metadata information is handled will need to be addressed though this is less relevant for the AI data center use case, where multicasting is less commonly used in the backend network. However, IFA offers significant improvements in network monitoring, such as tracking congestion points inside the fabric by using unique metadata and monitoring latency to the microsecond level of accuracy based on the copy of the original data packet.

Corrective Actions

We have covered different monitoring options available that can be used in the AI data center fabrics. As we get feed of data from the networking devices through any of the available mediums, the data can be stored on the servers. It is also important to understand how the data can be used:

- The data stored over a period of time can be used to derive the patterns.
- The patterns can be tagged with the corrective measures taken.
- Over a period of time, there may be different measures taken to address similar issues.
- The data can also be used to measure the effectiveness of the measures taken.

Currently, in most cases, humans are involved when any action needs to be taken. As the pattern identification and actions correlation matures, the monitoring platforms can be enabled to take the corrective action when patterns are identified. This will enable us towards more autonomous networks or self-driving networks. The use of artificial intelligence models has the potential to enable us to move towards it.

With AI data centers, the congestion occurs in a milli, micro and, in some cases, nano seconds. There are features being explored where the system takes decisions to handle these scenarios.

- It could be individual hop taking the decisions
- It could be the source of the stream taking the decisions by learning about congestion anywhere in the network.
- It could also be a monitoring platform, which is able to predict the recurrence of a known issue by looking at the pattern. And taking corrective measures for these.

It is also important to note that each node may be doing its best to deliver best network performance but as a system, the performance may not be good enough. Hence, end to end monitoring between the applications is being explored to achieve better results.

Some of the explorations are underway in UEC and covered in Chapter 12, "UEC - Ultra Ethernet Consortium." Also, there are exploration by the vendors as well as the large scale network deployment companies.

Summary

This chapter highlights key features of network monitoring. As you have seen, telemetry in-band and out-of-band capabilities have become the de facto standard in the AI/ML context. When dealing with much broader performance statistics, such as when using active probes to clone traffic or when using measurements of buffer occupation to analyze latency trends, it is important to use an analyzer server. For analysis of telemetry data, open-source systems are still relatively limited, and typically a specific vendor fabric manager is used to read the centrally collected data; the manager has a better chance of success at giving real-time analysis if the server also integrates an AI engine that is capable of gathering logic from different sources of data and presenting it to the end user in a simplified way. AI engine–based corrective actions are typically not yet integrated even in professional fabric managers and monitoring tools; human-based decisions are still currently involved to mitigate any issues that are reported by various monitoring tools, such as SNMP or telemetry.

Reference

Inband Flow Analyser (IFA): https://www.ietf.org/archive/id/draft-kumar-ippm-ifa-08.txt

Part 5: UEC – Ultra Ethernet Consortium

Chapter 12. Ultra Ethernet Consortium (UEC) [This content is currently in development.]

This content is currently in development.

Conclusion

Chapter 13. Scale-Up Systems [This content is currently in development.]

This content is currently in development.

Chapter 14. Conclusion [This content is currently in development.]

This content is currently in development.

Appendix A. Questions and Answers [This content is currently in development.]

This content is currently in development.

Appendix B. Acronyms

Acronym	Term	Short Definition
AALC	Air-assisted liquid cooling	A cooling method that uses air and liquid for data center equipment.
AEC	Active electrical cable	Copper cable with electronics to boost signal quality for short, high-speed connections.
AI	Artificial intelligence	Computer systems that perform tasks typically requiring human intelligence.
AIC	Application integration component	May refer to hardware or software integration modules.
AOC	Active optical cable	Fiber cable with built-in transceivers for high-speed data transmission.
API	Application programming interface	A set of rules for software components to communicate.
ASIC	Application-specific integrated Circuit	A chip designed for a specific application, not general use.
BCH	Bose–Chaudhuri–Hocquenghem	Error-correcting codes used in data transmission.
BGP	Border Gateway Protocol	The main routing protocol for exchanging information between networks on the Internet.
BTH	Base Transport Header	A header in InfiniBand and RoCE protocols for RDMA operations.
CCL	Collective communication library	Software for efficient data exchange between multiple computers or GPUs.
CNP	Congestion notification packet	A network message indicating congestion, used in RDMA networks.
CPU	Central processing unit	The main processor in a computer.

CQ	Completion queue	A data structure in RDMA for tracking completed operations.
CQE	Completion queue entry	An entry in a completion queue, indicating a finished RDMA operation.
DAC	Direct attach copper	Short copper cable for direct connection between network devices.
DC	Data center	A facility housing computer systems and networking equipment.
DCQCN	Data Center Quantized Congestion Notification	A congestion control protocol for RDMA over Ethernet.
DFE	Decision feedback equalization	A technique for reducing signal distortion in high-speed data links.
DLB	Dynamic load balancing	Adjusting network traffic distribution in real time for optimal performance.
DLB v2	Dynamic Load Balancing version 2	An enhanced version of DLB with more features.
DSCP	Differentiated Services Code Point	A field in IP headers for classifying network traffic.
DSP	Digital signal processor	A specialized processor for handling digital signals.
DWDM	Dense wavelength division multiplexing	A technology for sending multiple signals over a single fiber by using different wavelengths.

ECMP	Equal-cost multipathing	A routing strategy that uses multiple paths with equal cost for load balancing.
ECN	Explicit Congestion Notification	A network feature for signaling congestion without dropping packets.
EDR	Enhanced Data Rate	A high-speed InfiniBand link rate.
EoR	End of row	A data center network switch placement at the end of a row of racks.
EVPN	Ethernet Virtual Private Network	A technology for extending Layer 2 networks over Layer 3.
FAD	Flexible algorithm definition	A set of rules for routing algorithms in networks.
FC	Fibre Channel	A high-speed network technology for storage networks.
FCIP	Fibre Channel over IP	A technology that encapsulates Fibre Channel frames over IP networks.
FEC	Forward error correction	A method for detecting and correcting errors in data transmission.
FEP	Fabric endpoint	A node (server or switch port) in a network fabric.
FLOPS	Floating-point operations per second	A measure of computer performance, especially in scientific calculations.
GDS	GPUDirect Storage	Technology for direct data transfer between storage and GPU memory.
GLB	Global load balancing	Distributing network traffic across multiple paths or devices globally.
GPU	Graphics processing unit	A processor that is specialized for graphics and parallel computation.
GRH	Global Routing Header	A header in InfiniBand for routing packets between subnets.
HBA	Host bus adapter	A device that connects a server to storage or a network.

HPC	High-performance computing	Powerful computing systems for complex calculations.
HPE	Hewlett Packard Enterprise	A technology company.
IB	InfiniBand	A high-speed networking technology for data centers and HPC.
IBTA	InfiniBand Trade Association	The group that defines InfiniBand standards.
IC Req	Initialize Connection Request	A message to start a connection in NVMe over Fabrics.
INC	In-Network Collectives	A group of network switches performing collective operations (such as reduce and broadcast).
INT	In-band network telemetry	Real-time network monitoring that involves embedding telemetry in data packets.
IOPS	Input/output operations per second	A measure of storage system performance.
IP	Internet Protocol	The main protocol for sending data across networks.
iSCSI	Internet Small Computer Systems Interface	A protocol for linking data storage over IP networks.
JCT	Job completion time	The total time taken to finish a computing job.
KPI	Key performance indicator	A measurable value to evaluate success in a specific area.
LID	Local identifier	An address for InfiniBand devices within a subnet.
LLM	Large language model	An AI model trained on large text datasets (for example, GPT-4).
LLR	Link layer retry	A mechanism for retransmitting lost packets at the link layer.

LRH	Local Routing Header	A header in InfiniBand for local routing.
MAC	Media Access Control	The hardware address or layer responsible for network access.
MAC-VRF	MAC Virtual Routing and Forwarding	Virtualization of Layer 2 forwarding tables.
MIG	Multi-instance GPU	A GPU that is partitioned into multiple isolated instances.
ML	Machine learning	Algorithms that allow computers to learn from data.
MLPerf	Machine Learning Performance	A benchmark suite for AI/ML systems.
MMF	Multi-mode fiber	Optical fiber that supports multiple light paths for short distances.
MoR	Middle of row	Switch placement in the middle of a row of racks.
MTID	Multi-topology identifier	An identifier for different topologies in routing protocols.
NCCL	Nvidia Collective Communication Library	A library for multi-GPU and multi-node communication.
NDR	Next Data Rate	The latest high-speed InfiniBand link rate.
NFS	Network File System	A protocol for sharing files over a network.
NIC	Network interface card	Hardware that connects a computer to a network.
NNHN	Next-to-next-hop nodes	A BGP feature for advanced routing.
NRZ	Non-return-to-zero	A binary signal encoding method.

NVMe	Non-Volatile Memory Express	A protocol for fast storage access.
NVMe-oF	NVMe over Fabrics	The NVMe protocol extended over network fabrics.
NVMe-o-FC	NVMe over Fibre Channel	NVMe protocol run over Fibre Channel networks.
NVMe-o-RDMA	NVMe over RDMA	The NVMe protocol run over RDMA networks.
NVMe-o-TCP	NVMe over TCP/IP networks.	NVMe protocol run over standard TCP/IP networks.
NVSwitch	Nvidia Switch	An internal switch for GPU-to-GPU communication.
OAM	Operations, administration, and maintenance	Network management functions.
OFIWG	Open Fabric Interface Working Group	A group that is developing open network APIs.
OM1/2/3/4/5	Optical multimode fiber grades	Different types of multimode fiber for various speeds/distances.
OSFP	Octal small form-factor pluggable	A high-speed optical transceiver form factor.
OSPF	Open Shortest Path First	A routing protocol for IP networks.
PAM4	Pulse Amplitude Modulation 4-level	A signaling method for high-speed data.
PBR	Policy-based routing	Routing decisions based on policies, not just destination.
PCIe	Peripheral Component Interconnect Express	A high-speed interface for connecting hardware.

PDC	Packet Delivery Context	A logical session for packet delivery in UEC.
PDS	Packet Delivery Sublayer	The part of the UEC protocol that manages delivery semantics.
PDU	Protocol data unit	A unit of data at a particular layer of the network stack.
PFC	Priority Flow Control	A method used to pause specific traffic classes to prevent data loss.
POSIX	Portable Operating System Interface	A family of standards for maintaining compatibility between operating systems.
PRP	Physical region page	A pointer to memory in NVMe.
PUE	Power usage effectiveness	A metric for data center energy efficiency.
QDR	Quad Data Rate	An InfiniBand link speed.
QOS	Quality of service	Techniques for managing network traffic and performance.
QP	Queue pair	A pair of send/receive queues in RDMA or InfiniBand.
QSFP	Quad small form-factor pluggable	A compact, hot-pluggable transceiver for data communications.
QSFP-DD	Quad small form-factor pluggable double density	A higher-density version of QSFP for faster speeds.
RAG	Retrieval-augmented generation	An AI technique that combines retrieval and generation.

RCCC	Receiver credit congestion control	UEC's receiver-based congestion control.
RCCL	ROCm Communication Collectives Library	AMD's library for GPU communication.
RDMA	Remote direct memory access	Direct memory access from one computer to another without CPU involvement.
RETH	RDMA Extended Transport Header	A header in RDMA for extended transport information.
RFC	Request for comments	A type of publication from the IETF for standards and protocols.
RIFT	Routing in Fat Trees	A routing protocol for large-scale data centers.
ROD	Reliable ordered delivery	A UEC packet delivery mode that guarantees order and reliability.
RoCE	RDMA over Converged Ethernet	An RDMA protocol that is used in Ethernet networks.
RoCEv2	RDMA over Converged Ethernet version 2	RoCE using routable IP networks.
RT5	Route type 5	A type of route in EVPN for IP prefixes.
RUD	Reliable unordered delivery	A UEC packet delivery mode that guarantees reliability but not order.
RUDI	Reliable unordered delivery idempotent	A UEC mode for reliable, unordered, idempotent operations.

SACK	Selective acknowledgment	A TCP feature for acknowledging specific packets.
SAN	Storage area network	A dedicated network for storage devices.
SC	Subscriber connector	A type of fiber-optic connector.
SFP	Small form-factor pluggable	A compact, hot-pluggable transceiver.
SGL	Scatter gather list	A list that describing noncontiguous memory regions.
SID	Segment identifier	An identifier for a segment in segment routing.
SLB	Static load balancing	Fixed distribution of network traffic.
SMF	Single-mode fiber	Optical fiber for long-distance, single-light path transmission.
SNMP	Simple Network Management Protocol	A protocol for monitoring and managing network devices.
SR	Short reach	An optical transceiver for short distances.
SRv6	Segment Routing over IPv6	A routing technology that uses IPv6 extension headers.
SSD	Solid-state drive	A storage device that uses flash memory.
SUT	System under test	The system being evaluated or benchmarked.
TCAM	Ternary content-addressable memory	A type of memory for high-speed searching.
TCP	Transmission Control Protocol	A core protocol for reliable data transmission.

TE-LB	Traffic Engineering Load Balancing	Load balancing using traffic engineering techniques.
ToR	Top of rack	A network switch placed at the top of a server rack.
TSS	Transport security sublayer	The UEC's security layer for encrypted transport.
UDP	User Datagram Protocol	A connectionless network protocol for fast, unreliable transmission.
UEC	Ultra Ethernet Consortium	An industry group that defines next-generation Ethernet standards.
UET	Ultra Ethernet Transport	The new transport protocol defined by UEC.
UUD	Unreliable unordered delivery	A UEC packet delivery mode with no reliability or ordering guarantees.
VAST	—	A storage system company.
VNI	VXLAN network identifier	An identifier for VXLAN segments.
VOQ	Virtual output queue	A queuing method that prevents head-of-line blocking in switches.
VXLAN	Virtual Extensible LAN	A protocol for overlaying virtual networks over physical networks.
WDM	Wavelength-division multiplexing	Multiplexing multiple optical signals on a single fiber.
WQE	Work queue element	A descriptor for an RDMA operation.
WRED	Weighted random early detection	A congestion avoidance mechanism in networking.