DNN Training in Geo-Distributed Data Centers Using VXLAN and EVPN

M.Tech Thesis

by

Naved Inam

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY INDORE

May 2025

DNN Training in Geo-Distributed Data Centers Using VXLAN and EVPN

A THESIS

Submitted in partial fulfillment of the

requirements for the award of the degree

of

Master of Technology

by

Naved Inam

2302101010

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY INDORE

May 2025

INDIAN INSTITUTE OF TECHNOLOGY INDORE

CANDIDATE'S DECLARATION

I hereby certify that the work which is being presented in the thesis entitled **DNN Training in Geo-Distributed Data Centers Using VXLAN and EVPN** in the partial fulfillment of the requirements for the award of the degree of **Master of Technology** and submitted in the **Department of Computer Science and Engineering, Indian Institute of Technology Indore,** is an authentic record of my own work carried out during the period from July 2023 to May 2025 under the supervision of Dr. Sidharth Sharma, Indian Institute of Technology Indore, India.

The matter presented in this thesis has not been submitted by me for the award of any other degree of this or any other institute.

Naved Inam

19/May/2025
Signature of the Student with Date

(Naved Inam)

This is to certify that the above statement made by the candidate is correct to the best of my knowledge.

Sidharth

21/5/25
Signature of Thesis Supervisor with Date

(Dr. Sidharth Sharma)

Naved Inam has successfully given his M.Tech. Oral Examination held on **30/April/2025**.

Sidharth

Signature(s) of Supervisor(s) of M.Tech. thesis

Date: 23-5-25

Subhra Mazumdar

Signature of Chairman, PG Oral Board

Date: 23.05.2025

[illegible]

Signature of HoD

Date: 23 May 2025

ACKNOWLEDGEMENTS

I would like to express my heartfelt gratitude to my supervisor, **Dr. Sidharth Sharma**, Assistant Professor, Department of Computer Science and Engineering, IIT Indore, for his unwavering support, expert guidance, and constant encouragement throughout the course of my M.Tech thesis titled **DNN Training in Geo-Distributed Data Centers Using VXLAN and EVPN**. His deep knowledge, clear vision, and insightful feedback were invaluable at every stage of this research. I am truly thankful for the time and effort he dedicated to mentoring me and helping me grow as a researcher.

I am also grateful to **Mr. Gaurav Patel** a MS student at IIT Indore and **Mr. Mayank Ramnani** a PhD student at IIT Indore, for their generous support, constructive suggestions, and collaborative spirit during the technical implementation and experimentation phases of the project. Their input played a key role in overcoming several challenges throughout the work.

I would like to sincerely thank **Prof. Suhas S. Joshi**, Director, IIT Indore, for providing an excellent research environment and all the necessary resources that enabled me to carry out this work. His leadership and commitment to academic excellence have been truly inspiring.

I also extend my appreciation to all the faculty members of the CSE Department at IIT Indore for their valuable guidance, academic mentorship, and support during my M.Tech program. The knowledge and skills I gained through their courses and interactions have greatly contributed to the successful completion of this thesis.

Lastly, I am thankful to everyone who supported me, directly or indirectly, during this journey. Your encouragement and belief in me have been a great source of motivation.

Naved Inam

Master of Technology

Department of Computer Science and Engineering

Indian Institute of Technology Indore

Dedicated to My Family

ABSTRACT

The exponential growth of data and increasing demands for privacy and compliance drove the shift toward geo-distributed training of Deep Neural Networks (DNNs). Traditional centralized training in a single data center faced limitations such as WAN bandwidth constraints, high latency, and cross-border data restrictions. Conventional VLAN-based approaches also lacked scalability for multi-tenant, distributed environments due to limited ID space and poor isolation. To overcome these challenges, this thesis designed and evaluated a scalable, resilient, multi-tenant virtual network architecture for geo-distributed DNN training.

The system used VXLAN as a Layer 2 overlay and EVPN over MP-BGP as the control plane to extend network reachability across data centers while enabling isolated virtual networks. A realistic emulated environment was built using Containerlab and FRR, with a spine-leaf topology modeling inter-data center communication. The network was enhanced with Equal Cost Multi Path (ECMP) routing for load balancing and Bidirectional Forwarding Detection (BFD) for fast failure recovery. Prometheus, combined with SNMP and ping exporters, enabled continuous monitoring of link health, device uptime, and overall network connectivity.

Multi-tenancy was achieved using VXLAN Network Identifiers (VNIs), ensuring isolation of concurrent training workloads. Experiments confirmed intra-tenant communication and inter-tenant isolation, while parallel training using AllReduce and Parameter Server models validated support for distributed synchronization under emulated WAN conditions.

This work demonstrated that using VXLAN with EVPN enabled a resilient and scalable network architecture, well-suited for geo-distributed AI workloads, by supporting fault recovery, efficient routing, and strong isolation between tenants.

Contents

List of Figures

List of Tables

List of Abbreviations and Acronyms

DNN Deep Neural Network

VLAN Virtual Local Area Network

VXLAN Virtual Extensible Local Area Network

EVPN Ethernet Virtual Private Network

BGP Border Gateway Protocol

MP-BGP Multi Protocol Border Gateway Protocol

FRR Free Range Routing

ECMP Equal Cost Multi Path

BFD Bidirectional Forwarding Detection

SNMP Simple Network Management Protocol

VNI VXLAN Network Identifier

AI Artificial Intelligence

OSPF Open Shortest Path First

WAN Wide Area Network

MAC Media Access Control Address

IP Internet Protocol

MNIST	Modified National Institute of Standards and Technology
ML	Machine Learning
LAN	Local Area Network
IP	Internet Protocol
PS	Parameter Server
LLM	Large Language Model
UDP	User Datagram Protocol
VM	Virtual Machine
VTEP	VXLAN Tunnel End Point
RT	Route Target
RD	Route Distinguisher
ASN	Autonomous System Number
ARP	Address Resolution Protocol
AFI	Address Family Identifier
SAFI	Subsequent Address Family Identifier
NLRI	Network Layer Reachability Information
IMET	Inclusive Multicast Ethernet Tag
NLRI	Network Layer Reachability Information
RTT	Round Trip Time

Chapter 1

Introduction

Deep neural networks (DNNs) have become integral to the development of modern artificial intelligence systems, underpinning a broad array of applications such as image recognition, natural language processing, autonomous navigation, and scientific computing. These models are characterized by their computational intensity and high data throughput requirements, often necessitating large-scale, resource-rich environments for effective training. Traditionally, DNN training has been conducted within centralized data centers, where both the training data and computational infrastructure reside in a single location. While this approach has offered performance consistency and administrative simplicity, it is increasingly challenged by the evolving landscape of global data generation, regulatory compliance, and network constraints.

As data becomes more globally distributed and regulatory frameworks impose strict data sovereignty and privacy requirements, centralized training models face growing limitations. Organizations are often prohibited from aggregating data across borders due to legal restrictions such as the General Data Protection Regulation (GDPR) and other region-specific data protection laws. Furthermore, the reliance on wide-area network (WAN) links to transfer large datasets to a centralized facility introduces bandwidth bottlenecks, increased latency, and potential performance degradation. These factors render traditional centralized training strategies inadequate for many real-world AI workloads, particularly those involving federated or cross-region data sources.

To address these limitations, geo-distributed deep learning has emerged as a compelling alternative. This approach partitions training workloads and data across multiple geographically dispersed data centers, enabling model training to occur closer to the data source while complying with local regulatory requirements. Geo-distributed training offers benefits such as improved scalability, better fault isolation, and regulatory alignment. However, the shift from centralized to distributed training introduces new technical challenges, particularly in the domain of networking. Synchronization of model parameters across distributed nodes becomes significantly more complex and latency-sensitive. Network reliability and throughput directly impact model convergence and training stability, especially in synchronous training paradigms. In this context, the underlying network fabric becomes a critical enabler of performance and resilience.

Traditional Layer 2 technologies such as VLANs are not suited for such environments due to their limited scalability, inflexible configuration, and confinement to single-site broadcast domains. With support for only 4096 unique identifiers and a lack of native fault tolerance or tenant isolation mechanisms, these legacy solutions fail to meet the demands of dynamic, large-scale, multi-tenant machine learning workloads. In addition, conventional routing protocols like OSPF and standard BGP exhibit slow convergence times in the face of link or node failures, which can severely disrupt training pipelines, especially in long-running or tightly synchronized jobs.

This thesis proposes a modern, scalable, and fault-tolerant network architecture tailored to the requirements of geo-distributed DNN training. The proposed solution employs Virtual Extensible LAN (VXLAN) as the overlay tunneling protocol to enable Layer 2 connectivity across Layer 3 networks, significantly improving scalability and tenant isolation. VXLAN supports up to 16 million unique identifiers, allowing the infrastructure to accommodate numerous concurrent training workloads in a multi-tenant environment. Complementing VXLAN, the control plane leverages Ethernet VPN (EVPN) over Multiprotocol BGP (MP-BGP) to distribute MAC and IP reachability information across data centers efficiently and dynamically. This approach eliminates the need for manual configuration or flood-based learning, supports endpoint mobility, and improves overall network convergence.

To enhance performance and resiliency, the architecture integrates Equal-Cost Multi-Path (ECMP) routing for optimal traffic distribution and Bidirectional Forwarding Detection (BFD) for sub-second failure detection, and rapid recovery. ECMP enables the system to maximize bandwidth utilization across redundant paths, while BFD significantly reduces the time required to detect and respond to network failures, ensuring that training workloads remain uninterrupted. The implementation of this architecture is conducted in a simulated environment using ContainerLab and Free Range Routing (FRR), representing multiple virtual data centers connected via VXLAN tunnels in a spine-leaf topology. Routing and tunnel endpoint discovery are handled via MP-BGP, with full observability provided through Prometheus. Additional support from SNMP and Ping Exporters allows for comprehensive monitoring of latency, link performance, and route convergence.

To evaluate the effectiveness of the proposed solution, this thesis conducts a series of experiments designed to test the architecture's ability to support real-world DNN training under geo-distributed conditions. These include reachability tests, failure recovery benchmarks, ECMP path behavior analysis, multi-tenancy test and actual distributed training using the MNIST dataset. The experimental results demonstrate that the architecture achieves the desired goals of scalability, low-latency communication, rapid failover, and effective resource utilization, making it a viable networking foundation for distributed AI workloads.

By integrating modern networking protocols with open-source orchestration and monitoring tools, this thesis presents a comprehensive, scalable, and practical solution to the challenges posed by geo-distributed DNN training. The work not only highlights the limitations of conventional networking solutions in this context but also provides a validated architecture that can serve as a reference for future deployments in both academic and industrial settings. Through this contribution, the thesis aims to advance the field of distributed AI infrastructure and provide a foundation for more resilient and efficient machine learning systems operating at global scale.

Chapter 2

Related Work

Training deep neural networks (DNNs) in geographically distributed data centers involves a range of challenges stemming from limited wide-area network (WAN) bandwidth, increased latency, and the requirement to support multiple isolated tenants in a scalable manner. Many distributed machine learning (ML) systems rely on centralized aggregation approaches such as parameter servers or collective operations like AllReduce, which are well suited for high-speed local area networks (LANs). However, these architectures often experience performance bottlenecks when deployed in WAN environments. As a result, research efforts have focused on optimizing communication strategies and leveraging network virtualization techniques to improve the effectiveness of distributed training across multiple sites.

Several systems have been proposed to address WAN inefficiencies in ML training. Gaia introduced Approximate Synchronous Parallel (ASP), a synchronization model that reduces inter-data center communication by omitting insignificant updates while preserving model convergence [7]. Gaia improves bandwidth efficiency and training performance across geographically distributed sites by decoupling intra-data center communication from inter-data center synchronization. In a complementary direction, L3DML explores the use of programmable P4-based switches to perform in-network gradient aggregation [6]. This approach removes the need for a centralized parameter server and mitigates the impact of slow-performing data centers using rate-aware routing techniques. Other systems such as TicTac improve the efficiency of collective operations by optimizing the scheduling of AllReduce across WAN links [5], while [11] introduces a model that adapts to performance asymmetry across workers by aggregating updates in a non-blocking, partial manner.

Collaborative and decentralized training models have also emerged to reduce dependency on frequent communication and to support data locality or privacy constraints. The paper [14] demonstrates that combining local updates with periodic model averaging, supported by cyclical learning rates, can maintain convergence while significantly reducing synchronization overhead. Federated learning approaches such as FedAvg follow a similar principle, although they are primarily geared toward edge computing scenarios [10]. These strategies offer useful trade-offs in settings where full synchronization is impractical, even though they are less commonly used in data center-scale ML deployments.

To support distributed workloads in such environments, scalable virtual networking technologies have become essential. VXLAN enables Layer 2 extension over Layer 3 infrastructure and overcomes the limitations of VLANs by supporting a 24-bit identifier space for tenant segmentation [9]. However, VXLAN alone uses flood-and-learn behavior, which does not scale efficiently. EVPN addresses this limitation by introducing a control-plane MAC learning mechanism via MP-BGP, enabling dynamic endpoint discovery, reduced broadcast traffic, and seamless mobility [3, 13]. Practical implementations such as [4] illustrate how these overlays can be configured using FRRouting and container-based platforms to emulate multi-site data center fabrics.

These features are relevant for distributed ML, as they enable efficient routing, tenant separation, and flexible workload placement in virtualized infrastructures.

Workload migration is another important aspect in distributed ML systems, particularly for long-running jobs that may need to relocate due to load balancing or failure recovery. The paper [12] shows how MAC mobility and anycast gateways can be combined with proactive routing updates to support service continuity during migration between geographically distant sites. This helps preserve ongoing operations by ensuring that network state transitions do not disrupt connectivity or result in suboptimal routing paths.

Although substantial progress has been made in both machine learning system design and network virtualization, most studies evaluate these aspects independently. Machine learning experiments often assume ideal LAN conditions or use simplified WAN emulation without accounting for routing behaviors, convergence latency, or link failures. Likewise, network protocol evaluations typically rely on synthetic traffic and do not model the communication patterns of distributed training workloads. In this context, this thesis sets up a practical emulated environment using Containerlab, FRRouting, VXLAN, and EVPN to examine how network characteristics such as path convergence, link failure, and tenant isolation can influence distributed training performance. The focus is not on proposing new algorithms or systems, but rather on demonstrating, using existing open-source technologies, how distributed ML workloads behave in a controlled, geo-distributed network environment.

Chapter 3

Architectural Challenges and Protocol Selection

Training deep neural networks (DNNs) in geo-distributed environments introduces several challenges not present in traditional centralized or single-site setups. The combination of bandwidth constraints, routing complexity, fault tolerance needs, and multi-tenancy demands necessitates careful design choices in both the training architecture and the underlying network fabric. This section details the key architectural challenges encountered and provides a rationale for selecting specific protocols and technologies to address them.

3.1 Challenges in Geo-Distributed DNN Training

As machine learning workloads span multiple data centers, training systems must handle increased latency, limited bandwidth, and unstable network paths. These factors impact synchronization efficiency and overall throughput. Moreover, the need for isolation, fault recovery, and multi-tenant support adds further complexity. The following subsections outline the key challenges in such environments.

3.1.1 Synchronization Overhead in Distributed Training

In distributed deep learning, synchronization of model parameters is essential for convergence. In a parameter server (PS) architecture, workers periodically push their gradients to centralized servers and pull updated parameters for the next iteration. While effective in LAN settings, this approach becomes problematic in WAN environments, where network latencies are higher and bandwidth is more limited. Even moderate-scale DNNs (e.g., ResNet, Transformer) can involve hundreds of megabytes of model state that must be exchanged every few seconds.

AllReduce-based architectures, commonly used with frameworks like Horovod and NCCL, eliminate the central server by having peers directly exchange gradients. However, these architectures are highly sensitive to WAN-induced jitter and require consistent high-bandwidth links to maintain efficiency. As a result, both approaches face serious bottlenecks in geo-distributed settings unless the underlying communication fabric is optimized to handle inter-site traffic intelligently.

3.1.2 Scalability Limitations of Legacy VLAN Architectures

Traditional VLANs, based on IEEE 802.1Q, support a maximum of 4096 VLAN identifiers, limiting their applicability in modern large-scale and multi-tenant data centers. Additionally, VLANs operate strictly at Layer 2, which complicates routing across multiple physical sites or availability zones. Maintaining MAC address tables and broadcast domains at scale leads to flooding, loops, and convergence delays, making VLAN-based setups unsuitable for the dynamic and distributed nature of ML training workloads.

3.1.3 Fault Tolerance and Network Convergence Time

During DNN training, especially in long-running tasks such as large language model (LLM) training or real-time federated learning, the ability to quickly recover from node or link failures is critical. Traditional routing protocols such as OSPF or BGP can take several seconds to detect and recover from such events. In contrast, ML training pipelines expect fast failover to prevent wasted GPU cycles and inconsistent training states. Without sub-second convergence, failures during synchronization steps can lead to degraded performance or even training collapse.

3.1.4 Multi-Tenancy and Traffic Isolation

In multi-tenant environments where multiple users or applications train DNN models concurrently, isolation of traffic and compute resources is essential. The system must support thousands of simultaneous virtual networks, with isolation and minimal interference. Overlay networking is typically used to solve this, but it must be scalable, resilient, and compatible with modern control plane technologies.

3.2 Protocol Selection and Design Rationale

To address the above challenges, this thesis adopts a protocol stack composed of VXLAN, EVPN, MP-BGP, ECMP, and BFD, each selected based on its ability to meet the specific performance, scalability, and resiliency requirements of geo-distributed DNN training systems.

3.2.1 VXLAN for Layer 2 Overlays

Virtual Extensible LAN (VXLAN) is a Layer 2 overlay tunneling protocol that encapsulates Ethernet frames within UDP packets, allowing Layer 2 connectivity to be extended across Layer 3 networks [9]. VXLAN supports up to 16 million virtual network identifiers (VNIs), compared to only 4096 in VLANs, making it suitable for large-scale, multi-tenant cloud environments.

In this thesis, VXLAN is used to connect training nodes and parameter servers (or workers in AllReduce) across emulated data center boundaries, allowing the system to maintain logical proximity while being physically dispersed. By encapsulating traffic, VXLAN isolates tenant traffic and reduces the impact of broadcast and multicast storms.

3.2.2 EVPN as the Control Plane

Ethernet VPN (EVPN) serves as the control plane for VXLAN overlays. It replaces traditional flood-and-learn data plane mechanisms with control-plane-driven MAC/IP route advertisement via BGP [3, 13]. EVPN supports route types such as MAC/IP advertisement (Type 2), Inclusive Multicast (Type 3), and MAC mobility, allowing dynamic learning of endpoint reachability.

This design eliminates the need for static tunnels or multicast flooding, significantly improving convergence and scalability. EVPN also supports seamless VM mobility, critical when migrating workloads between sites by tracking MAC movement using sequence numbers in MAC mobility extended communities.

3.2.3 MP-BGP for Route Distribution

Multi-Protocol BGP (MP-BGP), as defined in [2], allows the exchange of routing information for multiple address families, including EVPN routes. In this implementation, MP-BGP distributes EVPN routes between leaf and spine nodes (VTEPs), enabling auto-discovery of remote peers and automatic tunnel creation.

MP-BGP also ensures that tenant MAC/IP information is propagated across the network, avoiding manual configuration and simplifying large-scale deployment. The use of Route Targets (RTs) and Route Distinguishers (RDs) enables flexible policy control and route segregation for different tenants or training pipelines.

3.2.4 ECMP for Load Balancing

Equal-Cost Multi-Path (ECMP)[1] routing is utilized to distribute training traffic across multiple parallel paths. This approach maximizes bandwidth utilization and enhances fault tolerance. Hash-based traffic distribution across ECMP paths helps prevent bottlenecks, especially in high-throughput scenarios such as synchronous parameter synchronization or gradient averaging.

Customized hashing policies (e.g., based on L3/L4 headers or inner payloads) ensure even distribution of flows and enable effective link utilization across the overlay fabric.

3.2.5 BFD for Rapid Failure Detection

Bidirectional Forwarding Detection (BFD) is a lightweight protocol that detects link or path failures in milliseconds, significantly faster than standard routing protocol timers [8]. BFD is integrated with BGP in this implementation to trigger fast convergence and path switchover when a link goes down.

This mechanism ensures that DNN training jobs remain connected with minimal disruption during network failures.

3.3 Summary

Geo-distributed DNN training introduces unique demands in terms of communication efficiency, scalability, resilience, and tenant isolation. Traditional L2 and routing protocols fall short under these conditions, necessitating a more modern and flexible networking stack.

This thesis adopts a robust protocol combination:

- VXLAN for scalable overlay tunneling [9].
- EVPN for MAC/IP control and mobility support [3, 13].
- MP-BGP for route advertisement and policy enforcement [2].
- ECMP for efficient path utilization [1].
- BFD for millisecond-scale failure detection and fast failover [8].

This architecture lays the foundation for the implementation described in the next section, which integrates these protocols in a simulated environment using open-source tools to evaluate performance under realistic constraints.

Chapter 4

Implementation

To evaluate the proposed architecture for geo-distributed DNN training, a complete emulated environment was built using open source tools such as Containerlab, FRRouting (FRR), and Docker. This setup models multiple interconnected data centers using a spine leaf topology, enabling the deployment of advanced routing protocols and overlay technologies like VXLAN and EVPN. The goal was to mimic real world conditions involving high latency, multi tenancy, and network failures to assess the robustness and scalability of the system. This chapter details the step by step implementation, starting with the network topology setup, followed by device configurations, protocol verifications, and the integration of a monitoring infrastructure for observability.

4.1 Topology Configuration and Deployment

To simulate a geo-distributed data center environment, we used Containerlab as the foundation for defining and deploying the network topology. Containerlab is a powerful tool that simplifies the creation of container-based network labs by allowing users to define complex topologies using a single YAML file. It abstracts away low-level container networking and lifecycle management, providing an efficient and reproducible way to emulate production-grade infrastructure.

The topology was defined using a YAML file that specified all the nodes, links, and bind mounts required for configuration. Each node was mapped to a container image, including FRRouting (FRR) routers for the control plane and Debian-based hosts for traffic generation and distributed training. Links were defined to simulate both intra-datacenter and inter-datacenter connectivity. Figure 4.1 shows an excerpt from the actual YAML topology file used in the deployment.

```
name: dc
mgmt:
  ipv4-subnet: 172.80.80.0/24
topology:
  defaults:
    kind: linux
    image: custom-debian:latest
  nodes:
    d1s1:
      mgmt-ipv4: 172.80.80.101
      image: custom-frr-snmpsupport:latest
    d1l1:
      mgmt-ipv4: 172.80.80.51
      image: custom-frr-snmpsupport:latest
    d1h1:
      mgmt-ipv4: 172.80.80.11
  links:
    - endpoints: ["d1s1:eth1", "d1l1:eth1"]
    - endpoints: ["d1l1:eth2", "d1h1:eth1"]
```

Figure 4.1: Example Containerlab YAML topology definition

The emulated topology followed a spine leaf architecture across two virtual data centers. Each data center contained two spine routers and three leaf routers. Hosts were connected to the leaf routers, and spine layers between the two data centers were linked to emulate WAN connectivity. This design allows the system to test ECMP, EVPN, and VXLAN behavior under realistic routing conditions. Figure 4.2 illustrates the complete topology used in the implementation.

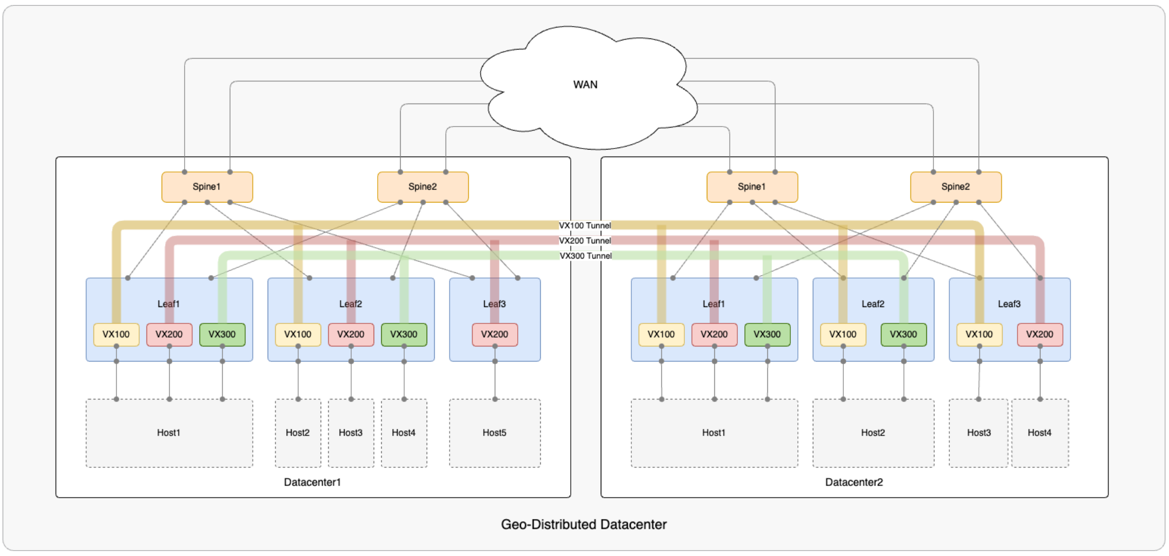


Figure 4.2: Emulated spine leaf topology used for VXLAN BGP EVPN deployment

After deploying the topology, all containers were up and running, interfaces were linked, and management IPs were reachable. A CLI screenshot confirming successful deployment is shown in Figure 4.3. At this point, the network was ready for further configuration.

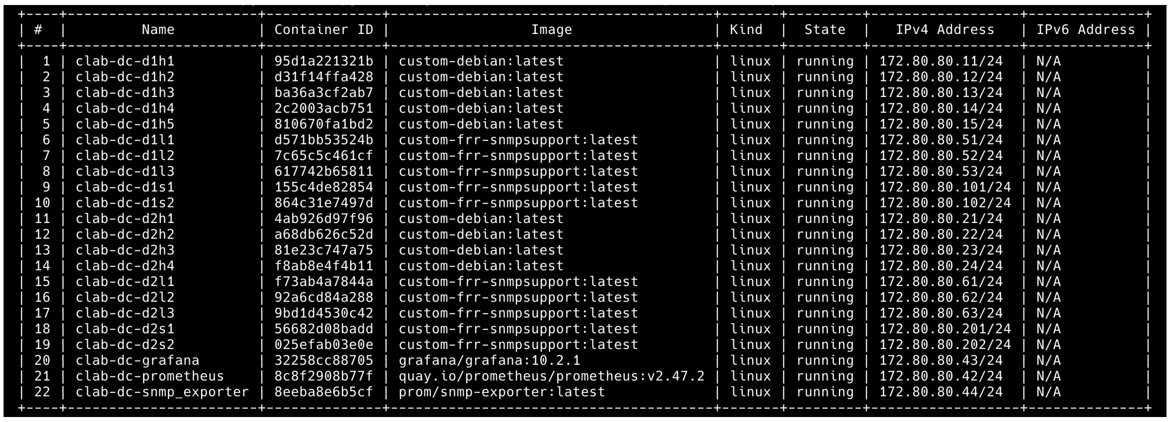

```
+----+-----------------------+--------------+---------------------------------------+-------+---------+------------------+--------------+
| #  |         Name          | Container ID |                 Image                 | Kind  |  State  |   IPv4 Address   | IPv6 Address |
+----+-----------------------+--------------+---------------------------------------+-------+---------+------------------+--------------+
|  1 | clab-dc-d1h1          | 95d1a221321b | custom-debian:latest                  | linux | running | 172.80.80.11/24  | N/A          |
|  2 | clab-dc-d1h2          | d31f14ffa428 | custom-debian:latest                  | linux | running | 172.80.80.12/24  | N/A          |
|  3 | clab-dc-d1h3          | ba36a3cf2ab7 | custom-debian:latest                  | linux | running | 172.80.80.13/24  | N/A          |
|  4 | clab-dc-d1h4          | 2c2003acb751 | custom-debian:latest                  | linux | running | 172.80.80.14/24  | N/A          |
|  5 | clab-dc-d1h5          | 810670fa1bd2 | custom-debian:latest                  | linux | running | 172.80.80.15/24  | N/A          |
|  6 | clab-dc-d1l1          | d571bb53524b | custom-frr-snmpsupport:latest         | linux | running | 172.80.80.51/24  | N/A          |
|  7 | clab-dc-d1l2          | 7c65c5c461cf | custom-frr-snmpsupport:latest         | linux | running | 172.80.80.52/24  | N/A          |
|  8 | clab-dc-d1l3          | 617742b65811 | custom-frr-snmpsupport:latest         | linux | running | 172.80.80.53/24  | N/A          |
|  9 | clab-dc-d1s1          | 155c4de82854 | custom-frr-snmpsupport:latest         | linux | running | 172.80.80.101/24 | N/A          |
| 10 | clab-dc-d1s2          | 864c31e7497d | custom-frr-snmpsupport:latest         | linux | running | 172.80.80.102/24 | N/A          |
| 11 | clab-dc-d2h1          | 4ab926d97f96 | custom-debian:latest                  | linux | running | 172.80.80.21/24  | N/A          |
| 12 | clab-dc-d2h2          | a68db626c52d | custom-debian:latest                  | linux | running | 172.80.80.22/24  | N/A          |
| 13 | clab-dc-d2h3          | 81e23c747a75 | custom-debian:latest                  | linux | running | 172.80.80.23/24  | N/A          |
| 14 | clab-dc-d2h4          | f8ab8e4f4b11 | custom-debian:latest                  | linux | running | 172.80.80.24/24  | N/A          |
| 15 | clab-dc-d2l1          | f73ab4a7844a | custom-frr-snmpsupport:latest         | linux | running | 172.80.80.61/24  | N/A          |
| 16 | clab-dc-d2l2          | 92a6cd84a288 | custom-frr-snmpsupport:latest         | linux | running | 172.80.80.62/24  | N/A          |
| 17 | clab-dc-d2l3          | 9bd1d4530c42 | custom-frr-snmpsupport:latest         | linux | running | 172.80.80.63/24  | N/A          |
| 18 | clab-dc-d2s1          | 56682d08badd | custom-frr-snmpsupport:latest         | linux | running | 172.80.80.201/24 | N/A          |
| 19 | clab-dc-d2s2          | 025efab03e0e | custom-frr-snmpsupport:latest         | linux | running | 172.80.80.202/24 | N/A          |
| 20 | clab-dc-grafana       | 32258cc88705 | grafana/grafana:10.2.1                | linux | running | 172.80.80.43/24  | N/A          |
| 21 | clab-dc-prometheus    | 8c8f2908b77f | quay.io/prometheus/prometheus:v2.47.2 | linux | running | 172.80.80.42/24  | N/A          |
| 22 | clab-dc-snmp_exporter | 8eeba8e6b5cf | prom/snmp-exporter:latest             | linux | running | 172.80.80.44/24  | N/A          |
+----+-----------------------+--------------+---------------------------------------+-------+---------+------------------+--------------+
```

Figure 4.3: Containerlab deployment output confirming successful startup of all nodes

4.2 Routers and Host Configuration

Once the topology was deployed, the next step involved configuring routing protocols and tunnel interfaces to enable seamless Layer 2 and Layer 3 communication across the virtual data centers. This section explains the configuration of BGP with EVPN, VXLAN Tunnel Endpoints (VTEPs), ECMP for load balancing, host network settings, and BFD for rapid failure detection.

4.2.1 BGP EVPN Configuration

Before deploying the topology, it was necessary to prepare the router configurations, with a particular focus on the setup required for BGP EVPN. These configurations were mounted into each router container using bind mounts defined in the Containerlab YAML file. This ensured that every router booted with its assigned routing logic and EVPN settings already in place.

A sample configuration for one of the leaf routers is shown in Figure 4.4. The router is configured with a unique BGP Autonomous System Number (ASN) and a router ID to identify it within the control plane. Peer groups are defined to simplify BGP neighbor configuration, particularly useful in spine-leaf topologies where each leaf peers with multiple spines. These peerings are established using directly connected interfaces, allowing BGP sessions to form automatically.

Within the configuration, the address-family ipv4 unicast block handles the exchange of traditional IP routes. The redistribute connected command ensures that directly connected subnets such as loopbacks are advertised into BGP.

```
interface lo
 ip address 1.1.10.1/32
exit

router bgp 65001
 bgp router-id 1.1.10.1
 bgp log-neighbor-changes
 bgp default l2vpn-evpn
 no bgp ebgp-requires-policy
 bgp bestpath as-path multipath-relax

 neighbor SPINE peer-group
 neighbor SPINE remote-as external
 neighbor eth1 interface peer-group SPINE
 neighbor eth2 interface peer-group SPINE

 address-family ipv4 unicast
  redistribute connected
 exit-address-family

 address-family l2vpn evpn
  neighbor SPINE activate
  advertise-all-vni
 exit-address-family
exit
```

Figure 4.4: Sample BGP-EVPN configuration for a leaf router in the emulated topology

More importantly, the address-family l2vpn evpn block enables the EVPN control plane. Here, EVPN is activated for the defined peer group, and the advertise-all-in command ensures that all locally configured VXLAN Network Identifiers (VNIs) are advertised through EVPN. This is essential for MAC learning across the overlay, as it allows the control plane to distribute MAC-to-VTEP mappings throughout the network. Without this, hosts attached to one leaf switch would not be discoverable by hosts connected elsewhere in the fabric.

4.2.2 VXLAN and Bridge Configuration

Once BGP and EVPN were configured and active, the next step was to set up the VXLAN Tunnel Endpoints (VTEPs) on the leaf routers. Each leaf node can host multiple VTEPs, one for each VNI, representing a separate Layer 2 segment. For every VNI, a dedicated VXLAN interface is created using the ip link add command. This interface is added to a Linux bridge, along with the host-facing physical interface. The bridge is then assigned a MAC and IP address so it can operate within the same subnet as the connected hosts.

Figure 4.5 shows a sample script used to configure a VTEP and its associated bridge. The VXLAN interface is created with a specific VNI, assigned a local source IP used as the tunnel endpoint, and connected to a bridge that links the host interface to the overlay. This setup allows the leaf to encapsulate outgoing traffic into VXLAN format and forward it to remote VTEPs, while also decapsulating incoming traffic addressed to local hosts. It also enables proper MAC address learning and forwarding throughout the fabric via EVPN.

```
ip link add vxlan100 type vxlan id 100 dstport 4789 local 1.1.10.1 nolearning ttl 5
brctl addbr bridge_v1d1h1
brctl addif bridge_v1d1h1 vxlan100
brctl stp bridge_v1d1h1 off
ip link set up dev bridge_v1d1h1
ip link set up dev vxlan100
brctl addif bridge_v1d1h1 eth3
ip link set bridge_v1d1h1 addr aa:bb:cc:01:01:01
ip addr add 192.168.1.101/24 dev bridge_v1d1h1
```

Figure 4.5: Sample VTEP and bridge configuration for a leaf router

After VTEPs are created and connected to the bridges, they start exchanging reachability information over the EVPN control plane. Each router advertises locally known MAC addresses, VNIs, and associated VTEP IPs using BGP. This allows every router in the network to learn how to reach hosts attached to other leaves, whether in the same data center or across the WAN to a different site. These advertisements form the basis for building and maintaining distributed Layer 2 connectivity over the Layer 3 underlay.

4.2.3 Host Configuration

After setting up EVPN and VXLAN, each host was manually configured with a unique MAC and IP address to maintain consistent identity across the system. Figure 4.6 shows a simple example of host-side interface configuration. Once the interface is brought up, the host initiates ARP to resolve IP-to-MAC mappings. The leaf router connected to that host observes these ARP packets and learns the MAC-to-IP binding. It then advertises this information to the rest of the fabric using EVPN. As a result, all routers in the overlay learn how to reach every host, making inter-host communication seamless across the distributed data center.

```
ip link set dev eth1 address aa:bb:01:02:01:01
ip addr add 192.168.1.3/24 dev eth1
ip link set eth1 up
```

Figure 4.6: Manual configuration of MAC and IP address for a host interface

4.2.4 ECMP and BFD Configuration

To enhance both fault tolerance and traffic distribution, ECMP and BFD were configured. ECMP was enabled using the `maximum-path` directive in the BGP configuration, allowing routers to install multiple equal-cost routes for the same destination. This helped distribute traffic across multiple uplinks in the spine-leaf fabric. Alongside ECMP, Bidirectional Forwarding Detection (BFD) was configured to enable fast failure detection between BGP peers. A profile named fast was created with transmit and receive intervals set to 100 milliseconds, and a detect multiplier of 3. This results in a failure detection time of approximately 300 milliseconds. The profile was applied to BGP neighbors using the bfd profile fast command to monitor peer liveliness efficiently.Figure 4.7 shows a combined configuration snippet for ECMP and BFD.

```
bfd
 profile fast
  transmit-interval 100
  receive-interval 100
  detect-multiplier 3
 exit
exit

router bgp 65001
 # previous settings
 neighbor eth1 bfd profile fast
 neighbor eth2 bfd profile fast
 # previous settings

 address-family ipv4 unicast
  # previous settings
  maximum-path 60
 exit-address-family

 # previous settings
exit
```

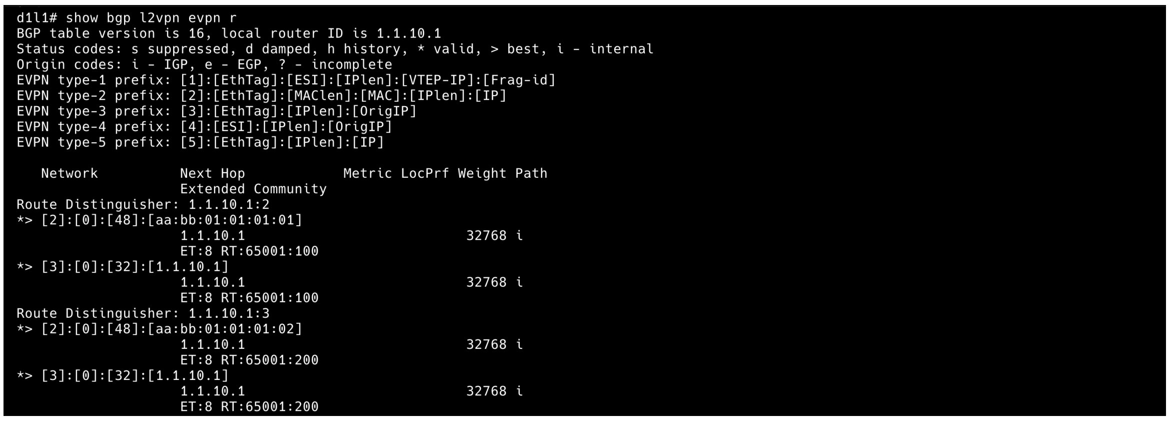
Figure 4.7: Combined ECMP and BFD configuration for a leaf router

With all routers and hosts configured, the network was fully functional and ready for protocol verification and experiment execution.

4.3 Protocol Verification

With the router and host configurations complete, the next step was to verify the correct operation of the control and data plane protocols across the emulated environment. This involved checking the behavior of EVPN advertisements, BGP session maintenance, MP-BGP route updates, VXLAN encapsulation, and BFD-based failure detection. A combination of FRR command-line utilities and Wireshark packet captures was used to perform this verification.

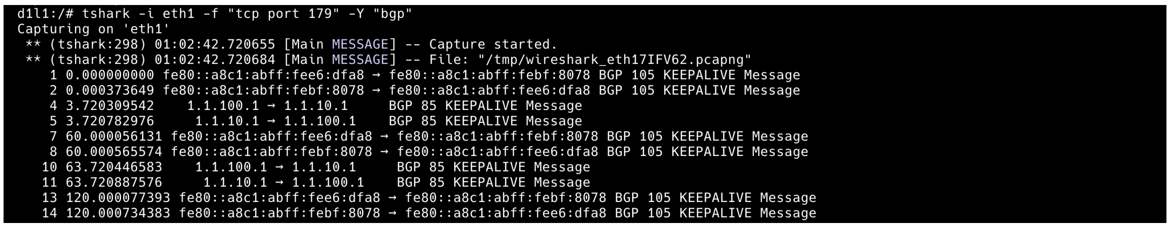
To confirm that EVPN was functioning as expected, the command `show bgp l2vpn evpn` was executed on the routers. This displayed both Route Type 2 and Route Type 3 advertisements. Route Type 2 contains MAC and IP information, allowing remote VTEPs to reach local hosts, while Route Type 3 advertises inclusive multicast groups required for broadcast and unknown unicast traffic forwarding. The successful exchange of both route types confirmed the operation of the EVPN control plane, as shown in Figure 4.8.



```
d1l1# show bgp l2vpn evpn r
BGP table version is 16, local router ID is 1.1.10.1
Status codes: s suppressed, d damped, h history, * valid, > best, i - internal
Origin codes: i - IGP, e - EGP, ? - incomplete
EVPN type-1 prefix: [1]:[EthTag]:[ESI]:[IPlen]:[VTEP-IP]:[Frag-id]
EVPN type-2 prefix: [2]:[EthTag]:[MAClen]:[MAC]:[IPlen]:[IP]
EVPN type-3 prefix: [3]:[EthTag]:[IPlen]:[OrigIP]
EVPN type-4 prefix: [4]:[ESI]:[IPlen]:[OrigIP]
EVPN type-5 prefix: [5]:[EthTag]:[IPlen]:[IP]

   Network          Next Hop            Metric LocPrf Weight Path
                    Extended Community
Route Distinguisher: 1.1.10.1:2
*> [2]:[0]:[48]:[aa:bb:01:01:01:01]
                    1.1.10.1                           32768 i
                    ET:8 RT:65001:100
*> [3]:[0]:[32]:[1.1.10.1]
                    1.1.10.1                           32768 i
                    ET:8 RT:65001:100
Route Distinguisher: 1.1.10.1:3
*> [2]:[0]:[48]:[aa:bb:01:01:01:02]
                    1.1.10.1                           32768 i
                    ET:8 RT:65001:200
*> [3]:[0]:[32]:[1.1.10.1]
                    1.1.10.1                           32768 i
                    ET:8 RT:65001:200
```

Figure 4.8: EVPN Route Type 2 and Route Type 3 advertisements

To verify that BGP sessions were being actively maintained, Wireshark was used to capture control plane traffic. Periodic keepalive messages were observed between BGP peers, indicating that the session state was stable. These messages, shown in Figure 4.9, serve to confirm the liveliness of the connection and are critical for maintaining long-lived peerings.

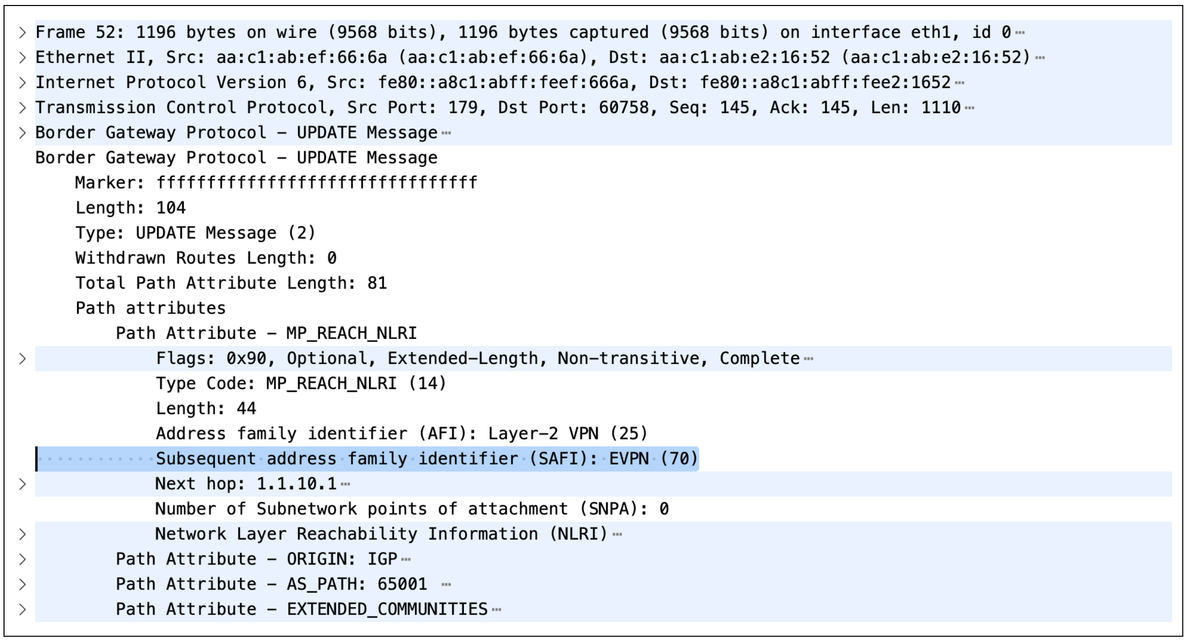


```
d1l1:/# tshark -i eth1 -f "tcp port 179" -Y "bgp"
Capturing on 'eth1'
 ** (tshark:298) 01:02:42.720655 [Main MESSAGE] -- Capture started.
 ** (tshark:298) 01:02:42.720684 [Main MESSAGE] -- File: "/tmp/wireshark_eth17IFV62.pcapng"
    1 0.000000000 fe80::a8c1:abff:fee6:dfa8 → fe80::a8c1:abff:febf:8078 BGP 105 KEEPALIVE Message
    2 0.000373649 fe80::a8c1:abff:febf:8078 → fe80::a8c1:abff:fee6:dfa8 BGP 105 KEEPALIVE Message
    4 3.720309542    1.1.100.1 → 1.1.10.1     BGP 85 KEEPALIVE Message
    5 3.720782976     1.1.10.1 → 1.1.100.1    BGP 85 KEEPALIVE Message
    7 60.000056131 fe80::a8c1:abff:fee6:dfa8 → fe80::a8c1:abff:febf:8078 BGP 105 KEEPALIVE Message
    8 60.000565574 fe80::a8c1:abff:febf:8078 → fe80::a8c1:abff:fee6:dfa8 BGP 105 KEEPALIVE Message
   10 63.720446583    1.1.100.1 → 1.1.10.1     BGP 85 KEEPALIVE Message
   11 63.720887576     1.1.10.1 → 1.1.100.1    BGP 85 KEEPALIVE Message
   13 120.000077393 fe80::a8c1:abff:fee6:dfa8 → fe80::a8c1:abff:febf:8078 BGP 105 KEEPALIVE Message
   14 120.000734383 fe80::a8c1:abff:febf:8078 → fe80::a8c1:abff:fee6:dfa8 BGP 105 KEEPALIVE Message
```

Figure 4.9: BGP keepalive message captured via Wireshark

MP-BGP update messages were also captured to confirm that EVPN-specific route advertisements were being propagated. These messages are identified by the use of Address Family Identifier (AFI) 25 and Subsequent Address Family Identifier (SAFI)

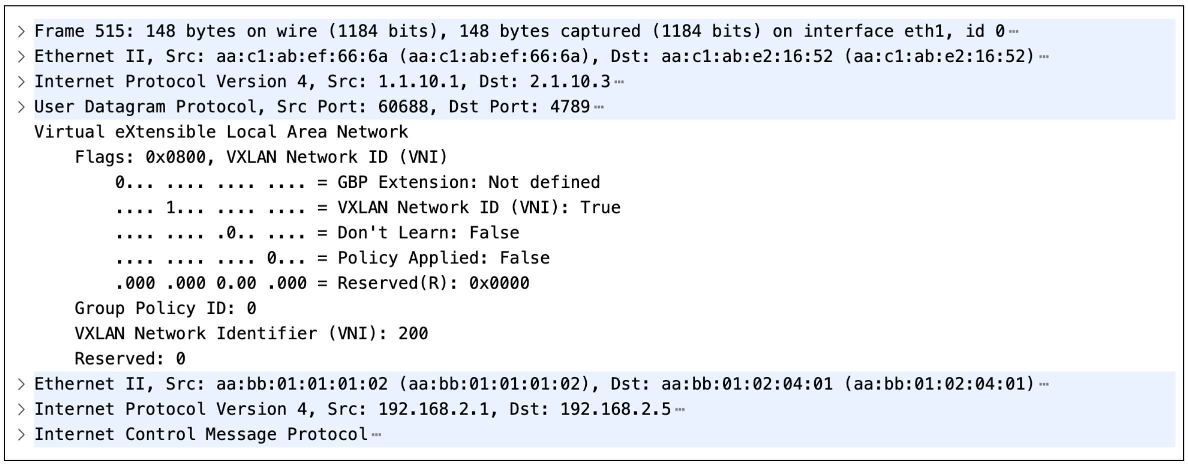
70. As shown in Figure 4.10, the update includes Network Layer Reachability Information (NLRI) with EVPN-specific attributes, allowing MAC and IP learning to occur through the control plane rather than flooding.



```
> Frame 52: 1196 bytes on wire (9568 bits), 1196 bytes captured (9568 bits) on interface eth1, id 0…
> Ethernet II, Src: aa:c1:ab:ef:66:6a (aa:c1:ab:ef:66:6a), Dst: aa:c1:ab:e2:16:52 (aa:c1:ab:e2:16:52)…
> Internet Protocol Version 6, Src: fe80::a8c1:abff:feef:666a, Dst: fe80::a8c1:abff:fee2:1652…
> Transmission Control Protocol, Src Port: 179, Dst Port: 60758, Seq: 145, Ack: 145, Len: 1110…
> Border Gateway Protocol - UPDATE Message…
  Border Gateway Protocol - UPDATE Message
      Marker: ffffffffffffffffffffffffffffffff
      Length: 104
      Type: UPDATE Message (2)
      Withdrawn Routes Length: 0
      Total Path Attribute Length: 81
      Path attributes
          Path Attribute - MP_REACH_NLRI
>             Flags: 0x90, Optional, Extended-Length, Non-transitive, Complete…
              Type Code: MP_REACH_NLRI (14)
              Length: 44
              Address family identifier (AFI): Layer-2 VPN (25)
              Subsequent address family identifier (SAFI): EVPN (70)
>             Next hop: 1.1.10.1…
              Number of Subnetwork points of attachment (SNPA): 0
>             Network Layer Reachability Information (NLRI)…
>         Path Attribute - ORIGIN: IGP…
>         Path Attribute - AS_PATH: 65001 …
>         Path Attribute - EXTENDED_COMMUNITIES…
```

Figure 4.10: MP-BGP EVPN update message captured via Wireshark

To ensure that the data plane was operating correctly, VXLAN traffic between hosts was analyzed. The captured packets showed a valid outer IP/UDP header, followed by the VXLAN header carrying the correct VNI, and the encapsulated Ethernet frame. This confirmed that the overlay network was properly encapsulating and decapsulating traffic across data center boundaries. A sample VXLAN packet is shown in Figure 4.11.



```
> Frame 515: 148 bytes on wire (1184 bits), 148 bytes captured (1184 bits) on interface eth1, id 0
> Ethernet II, Src: aa:c1:ab:ef:66:6a (aa:c1:ab:ef:66:6a), Dst: aa:c1:ab:e2:16:52 (aa:c1:ab:e2:16:52)
> Internet Protocol Version 4, Src: 1.1.10.1, Dst: 2.1.10.3
> User Datagram Protocol, Src Port: 60688, Dst Port: 4789
  Virtual eXtensible Local Area Network
      Flags: 0x0800, VXLAN Network ID (VNI)
          0... .... .... .... = GBP Extension: Not defined
          .... 1... .... .... = VXLAN Network ID (VNI): True
          .... .... .0.. .... = Don't Learn: False
          .... .... .... 0... = Policy Applied: False
          .000 .000 0.00 .000 = Reserved(R): 0x0000
      Group Policy ID: 0
      VXLAN Network Identifier (VNI): 200
      Reserved: 0
> Ethernet II, Src: aa:bb:01:01:01:02 (aa:bb:01:01:01:02), Dst: aa:bb:01:02:04:01 (aa:bb:01:02:04:01)
> Internet Protocol Version 4, Src: 192.168.2.1, Dst: 192.168.2.5
> Internet Control Message Protocol
```

Figure 4.11: VXLAN message captured via Wireshark

To further illustrate the flow of EVPN control and data plane operations, Figure 4.12 presents a step-by-step sequence of events that occur when a new host joins the network and begins communication with another remote host across the VXLAN-EVPN fabric. The sequence begins with Leaf1 sending an EVPN Type-3 route (Inclusive Multicast Ethernet Tag or IMET) through BGP to advertise its VTEP IP address for VNI 100. This is used by other VTEPs (e.g., Leaf2) to establish multicast group membership for unknown or broadcast traffic within the VXLAN segment.

Next, when Host1 joins the network and sends an ARP request, Leaf1 learns the MAC address (aa:bb:01:01:01:01) and associates it with VNI 100. It then advertises this information via an EVPN Type-2 route (MAC/IP advertisement), indicating that traffic for that MAC should be forwarded to its VTEP address (1.1.10.1). This update is reflected by the Spine node to all BGP peers, including Leaf2. Once Leaf2 receives this Type-2 route, it updates its forwarding table. When Host2 sends a packet to Host1, Leaf2 encapsulates the frame using VXLAN, targeting the destination VTEP (1.1.10.1). Upon arrival at Leaf1, the packet is decapsulated and delivered to Host1.

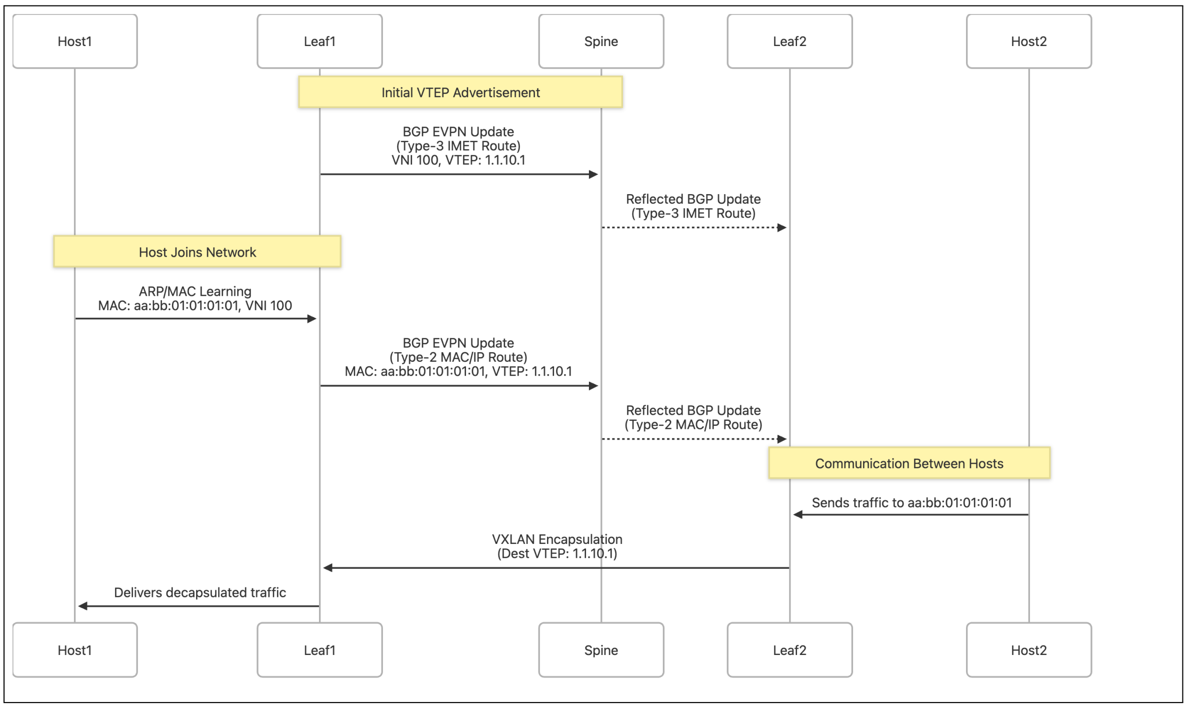
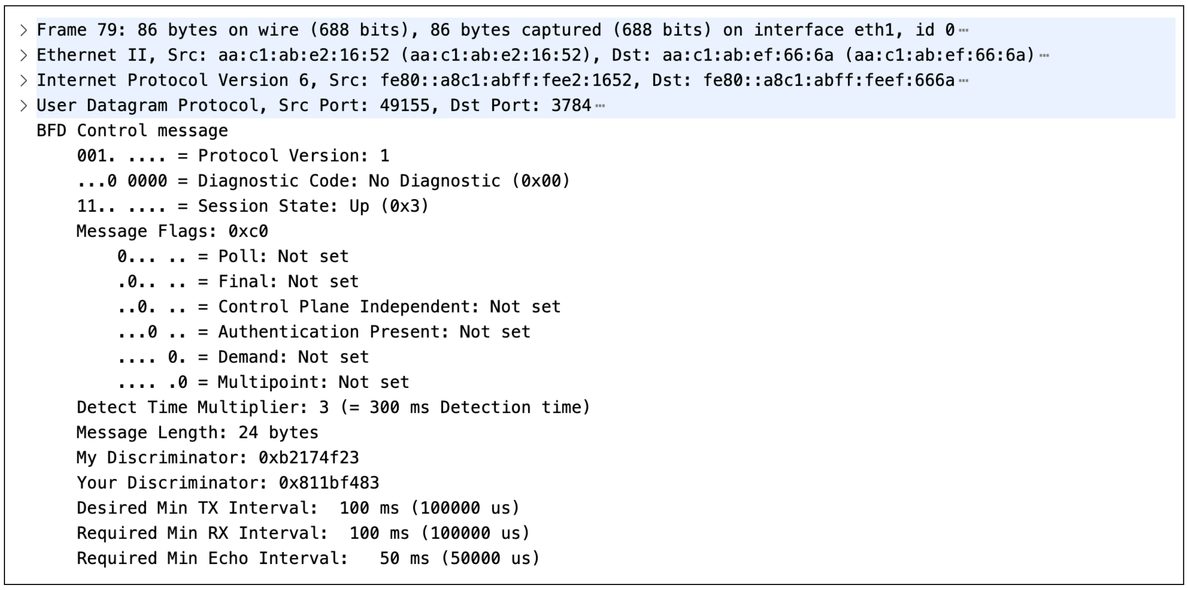


Figure 4.12: EVPN control and data plane operation: host join and inter-VTEP communication

Finally, BFD packets were captured to verify that fast failure detection was active. These lightweight control messages were exchanged between BGP peers at regular intervals over UDP port 3784. The capture confirmed that BFD was operating in the "Up" state with the configured timers in place, as shown in Figure 4.13. This confirmed the network's ability to detect failures rapidly and trigger path failover.



```
> Frame 79: 86 bytes on wire (688 bits), 86 bytes captured (688 bits) on interface eth1, id 0…
> Ethernet II, Src: aa:c1:ab:e2:16:52 (aa:c1:ab:e2:16:52), Dst: aa:c1:ab:ef:66:6a (aa:c1:ab:ef:66:6a)…
> Internet Protocol Version 6, Src: fe80::a8c1:abff:fee2:1652, Dst: fe80::a8c1:abff:feef:666a…
> User Datagram Protocol, Src Port: 49155, Dst Port: 3784…
  BFD Control message
      001. .... = Protocol Version: 1
      ...0 0000 = Diagnostic Code: No Diagnostic (0x00)
      11.. .... = Session State: Up (0x3)
      Message Flags: 0xc0
          0... .. = Poll: Not set
          .0.. .. = Final: Not set
          ..0. .. = Control Plane Independent: Not set
          ...0 .. = Authentication Present: Not set
          .... 0. = Demand: Not set
          .... .0 = Multipoint: Not set
      Detect Time Multiplier: 3 (= 300 ms Detection time)
      Message Length: 24 bytes
      My Discriminator: 0xb2174f23
      Your Discriminator: 0x811bf483
      Desired Min TX Interval:  100 ms (100000 us)
      Required Min RX Interval:  100 ms (100000 us)
      Required Min Echo Interval:   50 ms (50000 us)
```

Figure 4.13: BFD message captured via Wireshark

These protocol verifications confirmed that the underlying network components including EVPN for MAC and IP advertisement, BGP for control plane communication, VXLAN for overlay encapsulation, and BFD for rapid failure detection were correctly configured and operational. With this foundation in place, the network was fully prepared to support the distributed training experiments described in the subsequent sections.

4.4 Monitoring Infrastructure

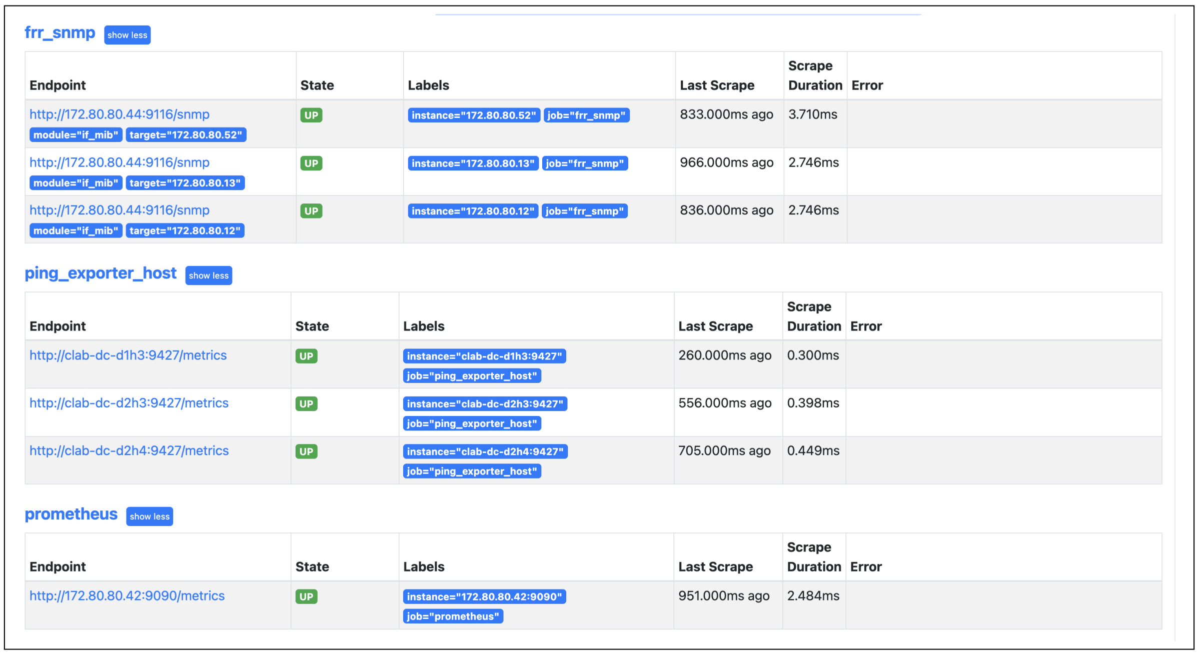
To ensure visibility into the health and performance of the emulated geo-distributed network, a monitoring stack was deployed using open source tools. Prometheus was selected as the central monitoring system, with SNMP and Ping Exporters installed on various nodes to expose metrics related to interface statistics, round trip time, and system availability. This setup enabled real-time tracking of network conditions and protocol behavior during training experiments.

Prometheus was configured using a YAML file that defined monitoring targets, including the IP addresses of routers and hosts. It periodically scraped metrics from each target using HTTP endpoints exposed by the exporters. Figure 4.14 shows a portion of the Prometheus configuration file that specifies SNMP and Ping Exporter endpoints.

```
global:
  scrape_interval: 1s

scrape_configs:
  - job_name: "ping_exporter_host"
    static_configs:
      - targets:
        - '192.168.1.1' # Replace with host node IPs or names in your network

  - job_name: "prometheus"
    static_configs:
      - targets: ["172.80.80.42:9090"]

  - job_name: "frr_snmp"
    metrics_path: /snmp
    static_configs:
      - targets:
          - '172.80.80.101' # Replace with host node IPs or names in your network

    params:
      module: [if_mib]
    relabel_configs:
      - source_labels: [__address__]
        target_label: __param_target
      - source_labels: [__param_target]
        target_label: instance
      - target_label: __address__
        replacement: 172.80.80.44:9116  # SNMP Exporter's IP and port
```

Figure 4.14: Prometheus configuration file with SNMP and Ping Exporter targets

Once configured, the Prometheus web interface confirmed that all targets were reachable and actively exporting metrics. Figure 4.15 shows the Prometheus Targets page, where each entry corresponds to a monitored router or host in the topology. A green status indicates successful metric collection, confirming that the exporters were running correctly.



frr_snmp show less

Endpoint	State	Labels	Last Scrape	Scrape Duration	Error
http://172.80.80.44:9116/snmp module="if_mib" target="172.80.80.52"	UP	instance="172.80.80.52" job="frr_snmp"	833.000ms ago	3.710ms	
http://172.80.80.44:9116/snmp module="if_mib" target="172.80.80.13"	UP	instance="172.80.80.13" job="frr_snmp"	966.000ms ago	2.746ms	
http://172.80.80.44:9116/snmp module="if_mib" target="172.80.80.12"	UP	instance="172.80.80.12" job="frr_snmp"	836.000ms ago	2.746ms	

ping_exporter_host show less

Endpoint	State	Labels	Last Scrape	Scrape Duration	Error
http://clab-dc-d1h3:9427/metrics	UP	instance="clab-dc-d1h3:9427" job="ping_exporter_host"	260.000ms ago	0.300ms	
http://clab-dc-d2h3:9427/metrics	UP	instance="clab-dc-d2h3:9427" job="ping_exporter_host"	556.000ms ago	0.398ms	
http://clab-dc-d2h4:9427/metrics	UP	instance="clab-dc-d2h4:9427" job="ping_exporter_host"	705.000ms ago	0.449ms	

prometheus show less

Endpoint	State	Labels	Last Scrape	Scrape Duration	Error
http://172.80.80.42:9090/metrics	UP	instance="172.80.80.42:9090" job="prometheus"	951.000ms ago	2.484ms	

Figure 4.15: Prometheus showing active monitoring targets

This monitoring setup provided key insights during the distributed training experiments, such as link availability, latency between data centers, interface usage, and protocol stability. By combining these metrics with packet captures and protocol logs, it was possible to correlate training performance with underlying network behavior. This observability was crucial in validating the impact of failure recovery mechanisms, ECMP behavior, and multi-tenant isolation in the experimental evaluations.

Chapter 5

Experiments

This chapter presents the experimental evaluation of the proposed geo-distributed DNN training system using VXLAN and EVPN. The experiments are structured to address key aspects of network performance and distributed training, including reachability, ECMP behavior, link failure recovery, multi-tenancy, and model training. All experiments were conducted on the emulated spine-leaf topology described in the implementation chapter, using Containerlab and FRRouting as the network substrate.

5.1 Reachability (Ping Test)

To verify end-to-end connectivity and establish baseline network performance, ICMP echo requests were sent from all hosts to all other hosts in the emulated topology. Figure 5.1 shows the round-trip time (RTT) measured when pinging from Datacenter1 Host1 to Datacenter2 Host1. In this initial setup, RTT values are extremely low, typically ranging from 20 to 200 microseconds. This is expected because all hosts are running as containers on the same physical machine, resulting in negligible propagation and switching delays.

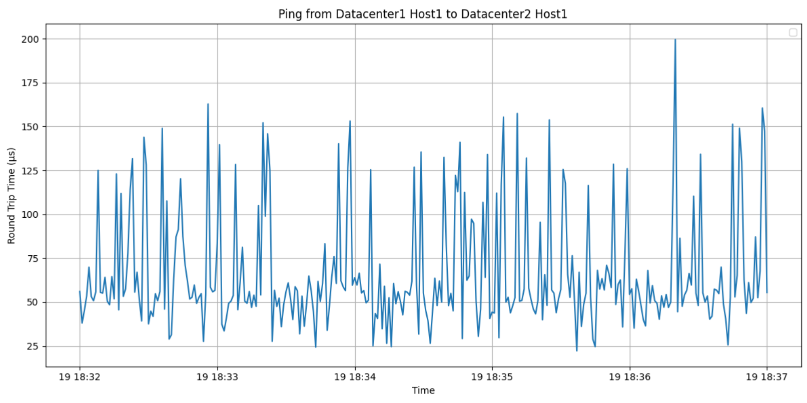


Figure 5.1: Ping RTT from Datacenter1 Host1 to Datacenter2 Host1 in the default environment.

However, these RTT values do not reflect those observed in real-world geo-distributed data centers, where WAN links introduce significant latency and jitter. To more accurately emulate real-world conditions, we used the Containerlab netem tool to introduce artificial delay and jitter on each inter-datacenter link. Specifically, a fixed delay of 5 ms and a jitter of 1 ms were applied per link. As shown in Figure 5.2, after applying these impairments, the RTT between Datacenter1 Host1 and Datacenter2 Host1 increased to approximately 50 ms, with observed variation consistent with the configured jitter.

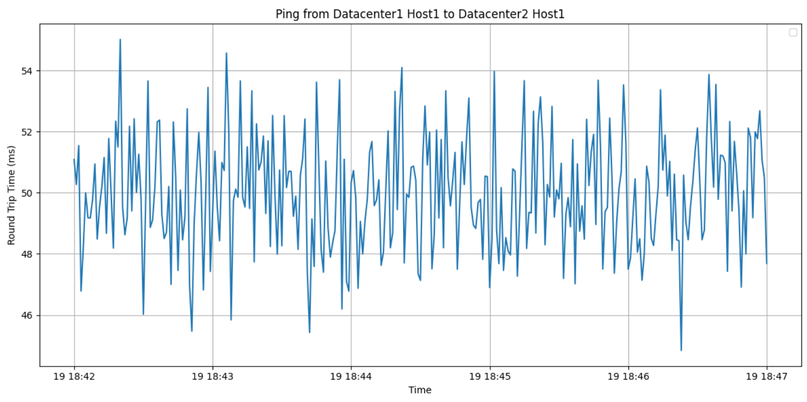


Figure 5.2: Ping RTT from Datacenter1 Host1 to Datacenter2 Host1 after introducing artificial delay and jitter using Containerlab's netem tool.

This approach ensures that the emulated network environment closely resembles the latency and variability expected in production geo-distributed deployments, providing a realistic basis for evaluating the performance of distributed DNN training and other network-dependent workloads.

5.2 Equal-Cost Multi-Path Routing (ECMP)

To evaluate the Equal-Cost Multi-Path (ECMP) routing, multiple traffic flows were generated from Datacenter1 Host1 to Datacenter2 Host1. As shown in the topology diagram (Figure 4.2), the goal was to observe how the network distributes traffic across parallel paths to improve throughput and resilience.

At the leaf node, incoming traffic from the host interface is split across two uplinks connecting to a different spine switch within the data center. This shows that ECMP effectively distributes the outgoing packets on both available spine paths, as shown in Figure 5.3.

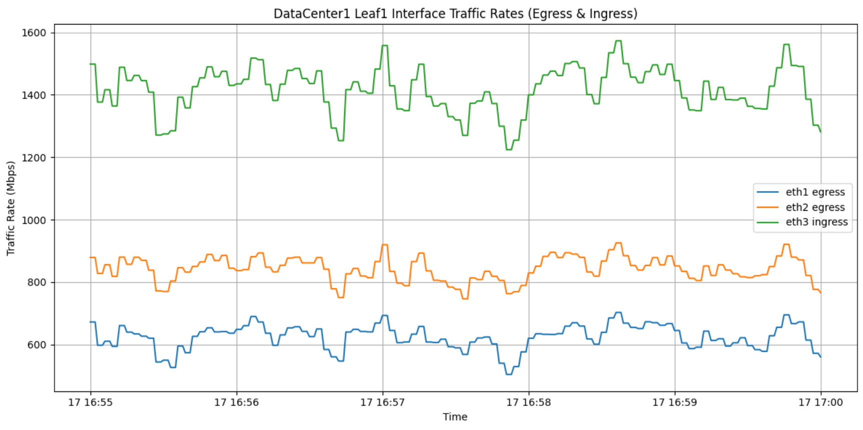


Figure 5.3: Traffic from Datacenter1 Host1 is distributed across both spine uplinks at the leaf node, demonstrating Equal-Cost Multi-Path (ECMP) routing.

On the spine node, the incoming traffic from the leaf is further distributed across two WAN-facing interfaces, each leading to a different spine in the remote data center. This confirms that ECMP continues to balance flows even at the spine layer, ensuring optimal utilization of all inter-datacenter links (see Figure 5.4).

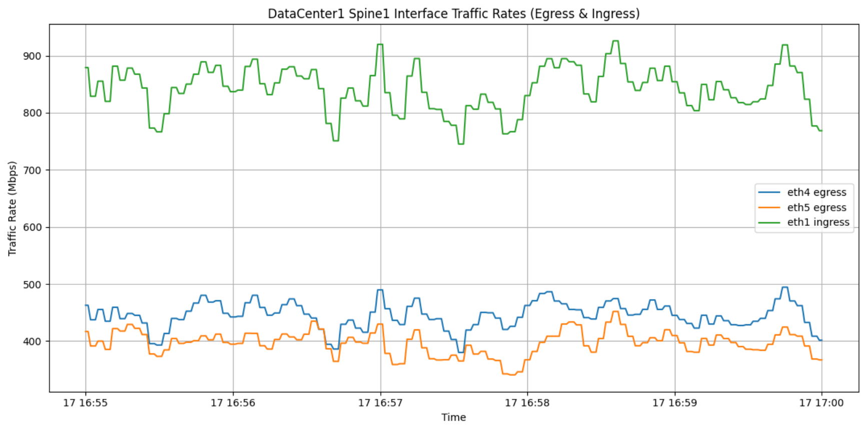
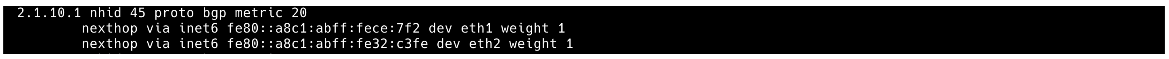


Figure 5.4: Traffic from leaf is distributed across two WAN links towards Datacenter2 at the spine node, confirming ECMP load balancing at the spine layer.

The ECMP behavior is controlled by a custom multipath hash policy (fib_multipath_hash_fields = 0x0DF7). This setting ensures that the hash function incorporates fields from both Layer 3 and Layer 4 headers, along with inner packet headers in the case of encapsulated traffic, thereby maximizing entropy and promoting balanced flow distribution across available paths. The routing table excerpt from a leaf node, as shown in Figure 5.5, illustrates multiple equal-cost next hops for the same destination prefix, confirming the deployment of ECMP.



```
 2.1.10.1 nhid 45 proto bgp metric 20
        nexthop via inet6 fe80::a8c1:abff:fece:7f2 dev eth1 weight 1
        nexthop via inet6 fe80::a8c1:abff:fe32:c3fe dev eth2 weight 1
```

Figure 5.5: Routing Table Excerpt Showing Multiple Equal-Cost Next Hops on a Leaf Node

These results validate that ECMP is functioning as intended, providing both path diversity and improved bandwidth utilization. This is critical for distributed DNN training workloads, where large volumes of data must be synchronized efficiently across geo-distributed sites.

5.3 Link Failure Recovery

To evaluate the network's resilience and convergence time during failures, we conducted a link failure recovery experiment under realistic WAN conditions. The test involved transferring data between two hosts with 50 ms latency and 5 ms jitter configured per link using the Containerlab netem tool. During continuous ping measurements, a critical path was disrupted by dropping all packets on the route, effectively emulating a link failure.

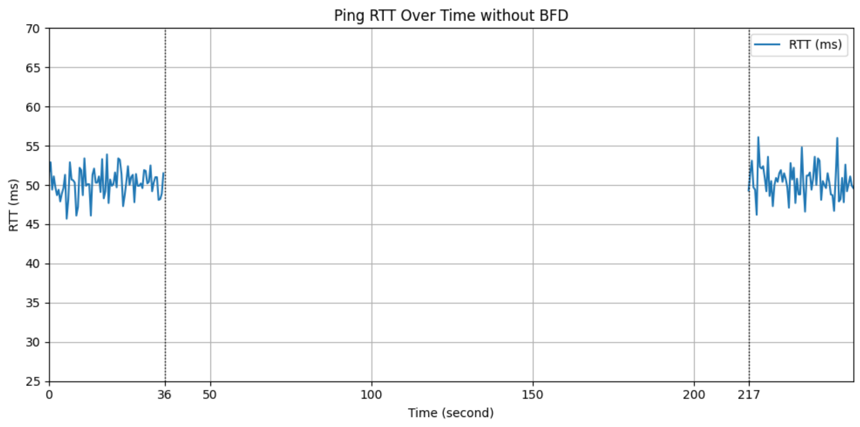


Figure 5.6: Ping RTT over time during link failure recovery using default BGP timers. Recovery takes approximately 180 seconds due to slow failure detection.

When only default BGP timers were used (keepalive interval of 60 seconds and hold timer of 180 seconds), the network required the full hold timer duration to detect the failure and converge to a new path. As shown in Figure 5.6, the RTT remains disrupted for approximately 180 seconds following the failure event. This prolonged convergence time is unsuitable for distributed training, as it can lead to significant computation loss and degraded performance.

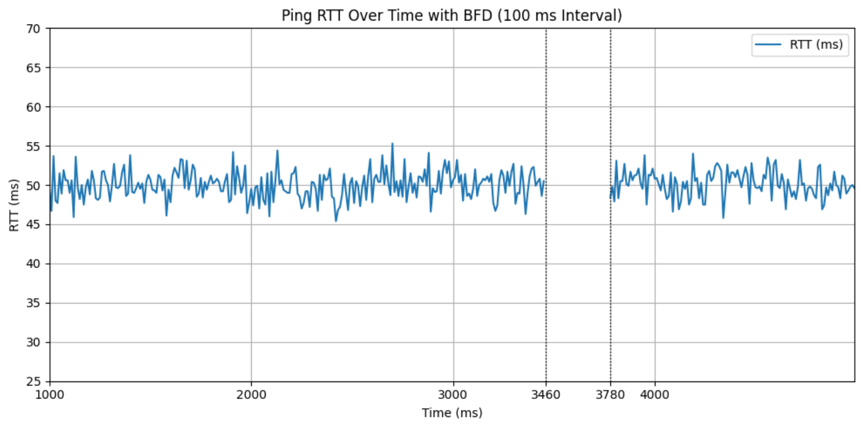


Figure 5.7: Ping RTT over time with BFD enabled (100 ms interval, 3 retries). Recovery is achieved in approximately 320 ms.

To enable rapid failure detection, Bidirectional Forwarding Detection (BFD) was configured between BGP peers. With BFD enabled at a 100 ms interval and 3 retries, the network detected the failure and restored connectivity in approximately 320 ms. Figure 5.7 illustrates the sharp drop and rapid recovery in RTT, demonstrating the effectiveness of BFD for fast convergence.

Further reducing the BFD interval to 10 ms (with 3 retries) allowed the network to detect and recover from the failure in about 110 ms, as depicted in Figure 5.8. This sub-second failover is essential for minimizing disruption during DNN training. These results demonstrate that integrating BFD with BGP-EVPN overlays dramatically reduces network convergence time following a link failure-from several minutes with default BGP timers to well under a second with aggressive BFD settings. This rapid failover capability is vital for sustaining high-performance and resilient distributed DNN training across geo-distributed data centers.

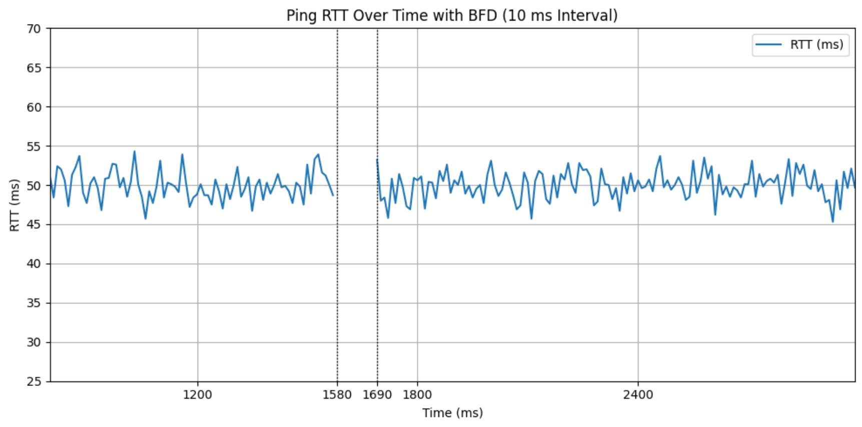


Figure 5.8: Ping RTT over time with BFD enabled (10 ms interval, 3 retries). Recovery is achieved in approximately 110 ms.

5.4 Multi-Tenancy

Multi-tenancy is a critical feature for modern data center networks, allowing multiple users, teams, or applications to securely share the same physical infrastructure while maintaining strict logical isolation. In this thesis, multitenancy is realized using VXLAN Network Identifiers (VNIs), with each tenant or training workload assigned a unique VNI. This ensures that traffic from different tenants remains isolated at the overlay level, regardless of the underlying physical topology.

The experimental setup, based on the spine-leaf topology described in Figure 4.2, was configured with multiple VNIs to emulate separate tenant environments. Hosts were assigned to different VNIs, and connectivity tests were conducted to verify both intra-tenant communication and inter-tenant isolation.

The results of the multitenancy experiment are summarized in Table 5.1. Hosts within the same VNI could communicate successfully, as shown by the low round-trip times between d1h1 and d2h1, both in VNI 100, and between d1h3 and d1h5, both in VNI 200. In contrast, attempts to communicate across VNIs, such as from d1h2 in

VNI 100 to d1h3 in VNI 200, or from d1h4 in VNI 300 to d2h4 in VNI 200, resulted in `destination host unreachable`, confirming that the VXLAN-EVPN overlay enforces strict tenant isolation.

Source Host	**Source VNI**	**Destination Host**	**Destination VNI**	**Result**
d1h1	100	d2h1	100	52.4 ms
d1h3	200	d1h5	200	38.2 ms
d1h2	100	d1h3	200	destination host unreachable
d1h4	300	d2h4	200	destination host unreachable

Table 5.1: Ping results between hosts in different VXLAN segments. Intra-tenant communication is successful with low RTT, while inter-tenant communication is blocked, confirming isolation.

These findings demonstrate that the VXLAN BGP-EVPN architecture, as implemented on the emulated spine-leaf topology, provides robust and scalable multi-tenant isolation. This capability is essential for supporting concurrent DNN training workloads and diverse user requirements in geo-distributed data center environments.

5.5 Distributed Training

To evaluate multi-tenancy and network performance for distributed deep learning, we conducted simultaneous training using both AllReduce and Parameter Server (PS) architectures over the emulated spine-leaf topology described previously. The network was configured with a 50 ms cross-datacenter latency and 5ms jitter to emulate realistic WAN conditions. Both training jobs used the MNIST dataset and the LeNet-5 model, but were mapped to different VXLAN segments to ensure isolation: AllReduce

(PyTorch) on VX300 and PS (MindSpore) on VX100. The topology, shown in Figure 5.9, includes both models running concurrently, with M1 denoting the PS setup and M2 denoting the AllReduce setup.

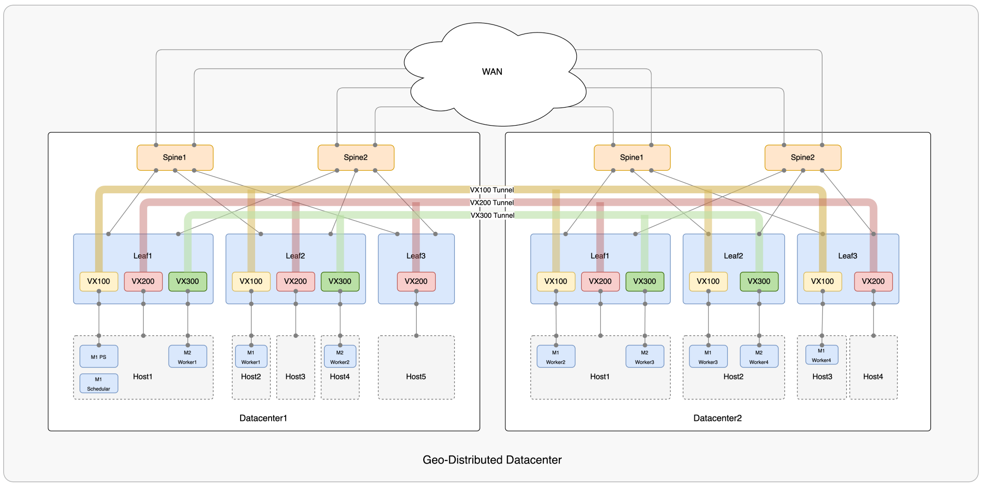


Figure 5.9: Experimental topology showing simultaneous deployment of Parameter Server (M1) and All Reduce (M2) architectures across the emulated geo-distributed data centers.

For AllReduce, four worker nodes participated in collective gradient synchronization using PyTorch's distributed library. The per-batch timing for gradient computation and synchronization exhibited significant variability, ranging from 350 ms to 1900 ms per batch, as shown in Figure 5.10. This variability reflects the sensitivity of decentralized synchronization to network conditions.

For the PS architecture, one parameter server (colocated with the scheduler) and four workers were used, all operating in VX100 and implemented with MindSpore. The batch-wise timing for gradient computation and synchronization was more stable, ranging from 500 ms to 900 ms, as shown in Figure 5.11. This stability is due to the centralized aggregation of gradients in the PS model.

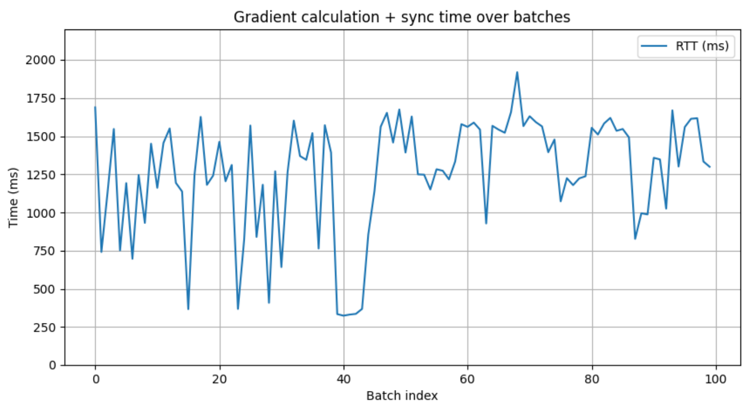


Figure 5.10: Batch-wise gradient computation and synchronization time for All Reduce (PyTorch).

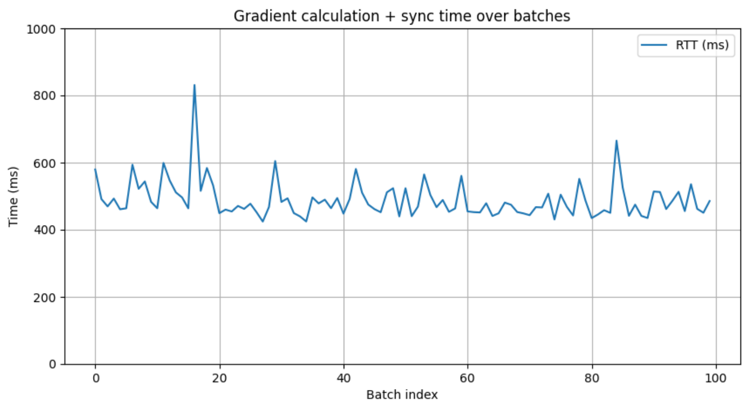


Figure 5.11: Batch-wise gradient computation and synchronization time for Parameter Server (MindSpore).

Running both jobs concurrently demonstrated the effectiveness of the VXLAN-EVPN network in supporting isolated, simultaneous distributed training workloads. The observed timings confirm that AllReduce is more sensitive to network conditions, while the PS model provides greater stability in per-batch training time. This experiment validates the multi-tenant capability and performance of the proposed virtualized network architecture under realistic geo-distributed training scenarios.

Chapter 6

Conclusion and Future Work

This thesis explored the design, implementation, and evaluation of a scalable and resilient networking infrastructure tailored for geo-distributed deep neural network training. In response to the evolving landscape of artificial intelligence, where data locality, regulatory compliance, and distributed compute resources increasingly shape system architecture, the work focused on overcoming the technical limitations inherent in legacy networking solutions. The primary goal was to ensure that geographically separated data centers could collaboratively train DNN models while maintaining high throughput, low latency, and strong fault tolerance.

To address this, a practical architecture based on VXLAN overlays and EVPN control plane mechanisms was developed and tested in a simulated environment. The solution was built using open-source tools such as ContainerLab and Free Range Routing (FRR), enabling the creation of a realistic multi-site data center topology. Through the deployment of VXLAN for scalable Layer 2 extension, EVPN for dynamic MAC/IP advertisement, and MP-BGP for route distribution, the infrastructure achieved seamless interconnection of training nodes while preserving network segmentation and tenant isolation.

The integration of Equal-Cost Multi-Path (ECMP) routing and Bidirectional Forwarding Detection (BFD) further enhances system performance and resilience. ECMP enabled efficient bandwidth utilization by distributing traffic across multiple parallel paths, while BFD significantly reduced failover times, ensuring continuity during transient network failures. These properties are especially crucial in distributed training scenarios where parameter synchronization is time sensitive and intolerant to network disruption.

Comprehensive testing validated the effectiveness of the proposed solution. Reachability and convergence tests confirmed the stability and reactivity of the network under dynamic conditions, while performance monitoring using Prometheus provided quantitative insights into traffic flow, latency, and fault recovery behavior. Importantly, the architecture successfully supported distributed training workloads, demonstrating its capability to serve as a functional backbone for real-world DNN training tasks.

The findings underscore the viability of using modern overlay and control plane protocols to meet the unique demands of geo-distributed AI training systems. Unlike traditional VLAN-based approaches, the proposed architecture offers a future-proof solution with the scalability, programmability, and agility required in geo-distributed infrastructures. It provides a foundational network blueprint upon which more complex and large-scale federated learning frameworks can be constructed.

While the current implementation was conducted within a controlled and simulated environment using ContainerLab, future work may focus on deploying the proposed architecture in production-grade clusters. Further evaluation with more computationally intensive models, such as ResNet-50 or Transformer-based architectures, would offer deeper insights into system performance under realistic AI training loads. Additionally, incorporating adaptive routing strategies and AI-driven traffic engineering could further enhance communication efficiency in dynamic, large-scale training scenarios.

In conclusion, this thesis contributes a practical and extensible network design tailored for modern AI workflows, bridging the gap between distributed machine learning requirements and the capabilities of contemporary networking protocols. It provides both a proof-of-concept and a roadmap for building resilient and scalable infrastructure in the age of data decentralization and cross-border AI collaboration.

Bibliography

[1] Cai, Y., Wei, L., Ou, H., Arya, V., and Jethwani, S. Protocol Independent Multicast Equal-Cost Multipath (ECMP) Redirect. RFC 6754, Oct. 2012.

[2] Chandra, R., Bates, T. J., Rekhter, Y., and Katz, D. Multiprotocol Extensions for BGP-4. RFC 4760, Jan. 2007.

[3] Drake, J., Henderickx, W., Sajassi, A., Aggarwal, R., Bitar, D. N. N., Isaac, A., and Uttaro, J. BGP MPLS-Based Ethernet VPN. RFC 7432, Feb. 2015.

[4] Elmadani, M., and Sati, S. O. Data center lab using vxlan data plane and bgp-evpn control plane. In *2023 4th International Conference on Data Analytics for Business and Industry (ICDABI)* (2023), pp. 354–358.

[5] Hashemi, S. H., Jyothi, S. A., and Campbell, R. H. Tictac: Accelerating distributed deep learning with communication scheduling, 2018.

[6] Hou, X., Gao, S., Liu, N., Yao, F., Lei, B., Zhang, H., and Das, S. L3dml: Facilitating geo-distributed machine learning in network layer. *IEEE Transactions on Network and Service Management PP* (01 2024), 1–1.

[7] Hsieh, K., Harlap, A., Vijaykumar, N., Konomis, D., Ganger, G. R., Gibbons, P. B., and Mutlu, O. Gaia: Geo-Distributed machine learning

approaching LAN speeds. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)* (Boston, MA, Mar. 2017), USENIX Association, pp. 629–647.

[8] Katz, D., and Ward, D. Bidirectional Forwarding Detection (BFD). RFC 5880, June 2010.

[9] Mahalingam, M., Dutt, D., Duda, K., Agarwal, P., Kreeger, L., Sridhar, T., Bursell, M., and Wright, C. Virtual eXtensible Local Area Network (VXLAN): A Framework for Overlaying Virtualized Layer 2 Networks over Layer 3 Networks. RFC 7348, Aug. 2014.

[10] McMahan, H. B., Moore, E., Ramage, D., Hampson, S., and y Arcas, B. A. Communication-efficient learning of deep networks from decentralized data, 2023.

[11] Miao, X., Nie, X., Shao, Y., Yang, Z., Jiang, J., Ma, L., and Cui, B. Heterogeneity-aware distributed machine learning training via partial reduce. In *Proceedings of the 2021 International Conference on Management of Data* (New York, NY, USA, 2021), SIGMOD '21, Association for Computing Machinery, p. 2262–2270.

[12] Noghani, K. A., Kassler, A., and Gopannan, P. S. Evpn/sdn assisted live vm migration between geo-distributed data centers. In *2018 4th IEEE Conference on Network Softwarization and Workshops (NetSoft)* (2018), pp. 105–113.

[13] Sajassi, A., Drake, J., Bitar, N., Shekhar, R., Uttaro, J., and Henderickx, W. A Network Virtualization Overlay Solution Using Ethernet VPN (EVPN). RFC 8365, Mar. 2018.

[14] Xu, K., Mi, H., Feng, D., Wang, H., Chen, C., Zheng, Z., and Lan, X. Collaborative deep learning across multiple data centers, 2018.