The bridge to possible

Cisco NX-OS Multi-Site VXLAN Fabric with RoCE Deployment Guide

Summary

The advent of Remote Direct Memory Access (RDMA) over converged ethernet on a hyper scale data center challenges DC network operator- to provide a low latency, lossless fabric at par with InfiniBand fabrics. Though lossless and low latency behavior can be achieved by allocating required network resources, those network resources are shared across other data center applications. The intent of this writing is to understand and configure RoCEv2 storage traffic to achieve lossless behavior over a VXLAN multi-site fabric.

Though this deployment guide describes the details of ROCE over VXLAN Multi-Site, one must be cautious about the geographic locations where VXLAN sites are deployed. We recommend that VXLAN sites are used only where the Data Center is spread over campuses or the same buildings. This helps achieve lossless converged ethernet constraints for ROCE traffic. This document provides details about multi-VXLAN and RoCE deployment.

To make the best decision, you should know the acceptable latency or RTT of the applications that are riding on the fabric. So, communication end nodes [Hosts or Targets] must be in proximity although they are deployed over Multi-Site VXLAN fabric.

This document covers,

1. RoCEv2 basics
2. Multi-Site VXLAN fabric and packet flow over the fabric
3. PFC and achieving lossless behavior with PFC
4. ECN and achieving lossless behavior on VXLAN fabric
5. Generating ECNs using WRED and Cisco's AFD QoS configs

Introduction

Non-Volatile Memory Express (NVMe) allows hosts to fully exploit the levels of parallelism possible with modern SSDs. As a result, NVMe reduces the I/O overhead and brings performance improvements relative to previous logical-device interfaces, including multiple long command queues, and reduced latency. SCSI and other previous interface protocols were developed for use with slower hard disk drives where a lengthy delay, relative to CPU operations, exists between a request and data transfer, where data speeds are slower than RAM speeds, and where disk rotation and seek time that is led to further optimization requirements.

NVMe over Fabrics (NVMe-oF)

The NVMe protocol is more than just connecting a local flash drive inside a server, it may also be used over a network. When used in this context, a network "fabric" enables any-to-any connections among storage and server elements. NVMe over Fabrics (NVMe-oF) enables a high-performance storage network with latency that rivals direct attached storage. As a result, fast storage devices can be shared.

NVMe/RDMA – Supported on InfiniBand or Ethernet networks

An InfiniBand (IB) architecture provides credit-based flow control for lossless, TCP/IP bypass, RDMA, and zero-copy. The first version of RDMA over Converged Ethernet (RoCE), supports zero-copy like InfiniBand by enabling the hardware network adapter to copy the data directly from the application memory. To maintain a reliable connection, the destination Queue Pair (QP) maintains the packet flow using the sequence number in the IB Base Transport Header (BTH). As there is no IP information on RoCEv1, the lossless behavior can only be achieved using Priority Flow Control (PFC) in the network.

What is RDMA?

Direct Memory Access (DMA) is the ability of a device to access host memory directly, without the intervention of the CPU. Remote Direct Memory Access (RDMA) is the ability to access (read and write) memory on a remote machine without interrupting CPU operations.

RDMA Main Advantages

- Zero-copy – Applications can perform data transfers without engaging the network software stack. Data is sent and received directly to the buffers without being copied between the network layers.
- Kernel bypass – Applications can perform data transfers directly from user-space without kernel involvement
- No CPU involvement – Applications can access remote memory without consuming any CPU time in the remote server. The remote memory server will be read without any intervention from the remote process (or processor). Moreover, the caches of the remote CPU will not be filled with the accessed memory content

What is RoCE?

RDMA over Converged Ethernet (RoCE – pronounced "Rocky") is a network protocol that allows Remote Direct Memory Access (RDMA) over an Ethernet network. It does this by encapsulation of an InfiniBand transport packet over Ethernet. There are two versions of RoCE:

- RoCEv1 – Ethernet link layer protocol (Ethertype 0x8915) allows communication between two hosts in the same Ethernet broadcast domain. So, layer-2 only that is not routable.
- RoCEv2 – Enhances RoCEv1 with a UDP/IP (IPv4 or IPv6) header and adds layer-3 routability. RoCEv2 is also known as Routable RoCE.

Ethernet L2 Header	Ether Type	IB GRH	IB BTH + L4 Header	IB Payload	ICRC	FCS

Figure 1. RoCEv1 Packet Format

Later, RoCE is carried over IP and UDP which leads to RoCEv2. The UDP destination port-4791 indicates that the payload is an InfiniBand payload with InfiniBand Base Transport header. RoCEv2 can use different source UDP ports for different QPs that help ECMP load sharing. RoCEv2 is used on an IP fabric where the lossless nature is achieved using Priority Flow Control (PFC) or using Explicit Congestion Notification (ECN) on a lossy IP fabric.

RoCEv2 on a lossy IP fabric is called Resilient RoCE, uses ECN bits on the IP header to achieve lossless behavior during congestion in the network.

Ethernet L2 Header	Ether Type	IP Header	UDP Header	IB BTH +L4 Header	IB Payload	ICRC	FCS

Figure 2. RoCEv2 Packet Format

On a storage network, we have servers that do read or write operations with a storage device. The servers are referred to as Initiators and the storage devices are called Targets.

How to support RoCEv2 on the network

A RoCEv2 network fabric should use various intelligent congestion control technologies to eliminate the potential packet loss and high latency of a traditional Ethernet network. The goal is to have zero-packet-loss, low-latency, and high-throughput network for RoCEv2 distributed applications, meeting the stringent performance requirements of these applications

The benefits of introducing RoCE in data center infrastructure include:

- Lower cost of ownership because a separate storage networking infrastructure is not needed.
- Higher ROI across traditional and modern agile infrastructures.
- Improved overall CPU utilization for using applications.
- Efficient host memory usage.
- Higher throughput and lower latency for computing and storage traffic.

In our case study with VXLAN Multi-Site Fabric, the lossless behavior is achieved using Priority Flow Control (PFC) and Explicit Congestion Notification (ECN).

Priority Flow Control (PFC)

PFC, which is also referred to as Class-based Flow Control (CBFC) or Per Priority Pause (PPP), is a mechanism that prevents frame loss caused due to congestion. PFC is like 802.3x Flow Control (pause frames) or link-level flow control (LFC). However, PFC functions on a per class-of-service (CoS) basis: When a buffer threshold is exceeded due to congestion, 802.3x flow control or LFC sends a pause frame to its peer to pause all data transmission on the link for a specified period. When the congestion is mitigated, when traffic comes under the configured threshold, a resume frame is generated to restart data transmission on the link.

In contrast, during congestion, PFC sends a pause frame that indicates which CoS value needs to be paused. A PFC pause frame contains a 2-octet timer value for each CoS that indicates the length of time that the traffic must be paused. The unit of time for the timer is specified in pause quanta. A quanta is the time required for transmitting 512 bits at the speed of the port. The range is from 0 to 65535. A pause frame with a pause quanta of 0 indicates a resume frame to restart the paused traffic. PFC asks the peer to stop sending frames of a particular CoS value by sending a pause frame to a well-known multicast address. This pause frame is a one-hop frame that is not forwarded when received by the peer. When the congestion is mitigated, PFC can request the peer to restart transmitting frames

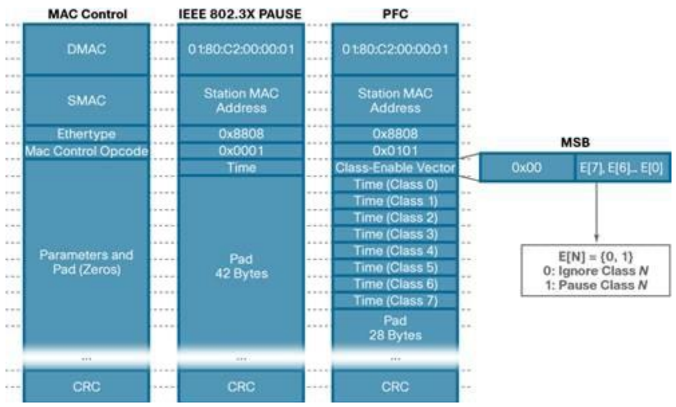


Figure 3. IEEE 802.3x PAUSE and PFC Frame Format

PFC Storm

A malfunctioning NIC on a host may not be able to receive any traffic from the network and continues sending PFC pause frames towards the switch. Lossless switch paths do not drop packets but decline to receive more packets when their buffers fill up. If the end-port queue is stuck for a long time, the buffers fill up not only for the target switch but also on all switches with problematic port queues in the traffic forwarding path. This leads to endless PFC pause frames, also called a PFC storm, being observed on all switch ports along the path to the traffic source.

PFC Watchdog (PFCWD) on Nexus 9000 Series Switches

To mitigate this, the PFC watchdog can be used on the Nexus switches to prevent congestion. When the switches detect this situation on any egress queue, all the packets in the queue are flushed, and new packets that are destined to the same queue are dropped as well until PFC storming is relieved. We lose some packets with this solution, but only temporarily. Without this mechanism, the traffic of this queue is blocked all along the path, until the faulty NIC is restarted or replaced.

Explicit Congestion Notification (ECN)

ECN is a notification mechanism on the packet forwarding direction which marks packets instead of dropping on a WRED-enabled queue when the average queue length exceeds a specific threshold value. When configured with the WRED ECN feature, the Nexus 9k switches mark the ECN bit at the point of congestion. The end hosts' application on seeing the ECN bit enabled packets ask the source traffic to slow down by an application-specific slow down mechanism.

In case of congestion, the network device marks the packet with ECN Congestion Encountered bit to (0x11). This ECN flag set packet arrives at the destination and the destination sends a notification to the sender to reduce the traffic rate.

ECN uses the two least significant (right-most) bits of the Traffic Class field in the IPv4 or IPv6 header to encode four different code points:

```
0x00 – Non-ECN-Capable Transport (Non-ECT)

0x10 – ECN Capable Transport 0 (ECT-0)

0x01 – ECN Capable Transport 1 (ECT-1)

0x11 – Congestion Encountered (CE)
```

Using PFC and ECN together

For optimal RDMA performance in rapidly changing and dynamic network environments, both PFC and ECN can be used together. In that case, congestion caused by traffic patterns such as in-cast can be easily mitigated with ECN, because capabilities that exist anywhere in the data path congestion are signaled to the endpoints. However, if congestion is experienced close to the endpoints and caused by a bursty application by the sender, PFC efficiently mitigates and manages the congestion by slowing down the traffic rate from the sender. In summary, ECN and PFC are used in RoCEv2 traffic to provide congestion control and to ensure that critical traffic is not delayed or lost due to congestion. These mechanisms work together to allow for a more efficient use of network resources and to provide a more predictable and consistent performance for RoCEv2 traffic.

RoCE can also be deployed using just PFC or—with advanced RoCE adapters, from few vendors—with just ECN. Using only PFC or ECN, performance for ordinary applications can be satisfied, but highest performance is usually achieved only by coupling PFC and ECN.

Multi-Site VXLAN Fabric

VXLAN fabric is a network virtualization technology that enables the creation of multiple virtual domains within a single physical network that can span multiple physical locations. This is achieved by extending the local VLAN resources across the fabric by mapping VLAN to VXLAN VNIs. The network routers in this fabric which are referred as VXLAN Tunnel Endpoints (VTEPs), connect different sites by maintaining a VXLAN Routing and Forwarding Instance (VRF) table on each VTEP to forward the traffic. This feature allows consolidation of network resources, improved security, and flexibility in managing virtual machines across multiple sites, resulting in a more efficient and reliable network infrastructure.

Multi-Site VXLAN fabric is a collection of multiple such VXLAN fabrics maintaining the same segmentation and isolation across geographical locations. This is facilitated by Border Gateway Routers (BGW) along with Data Center Interconnect (DCI) routers.

Main components of the EVPN Multi-Site architecture

The main functional component of the EVPN Multi-Site architecture is the border gateway, or BGW. BGWs separate the fabric-side (site-internal fabric) from the network that interconnects the sites (site-external DCI) and mask the site-internal VTEPs.

Commonly, an EVPN Multi-Site deployment consists of two or more sites, which are interconnected through a VXLAN BGP EVPN Layer 2 and Layer 3 overlay (Figure 4). In this scenario, the BGW is connected to the site-internal VTEPs (usually through spine nodes) and to a site-external transport network that allows traffic to reach the BGWs at other, remote sites. The BGWs at the remote sites have site internal VTEPs behind them. Only the underlay IP addresses of the BGWs are seen inside the transport network between the BGWs. The site internal VTEPs are always masked behind the BGWs.

For more information related to multi-site VXLAN, see VXLAN EVPN Multi-Site Design and Deployment White Paper.

VXLAN Multi-Site Control Plane

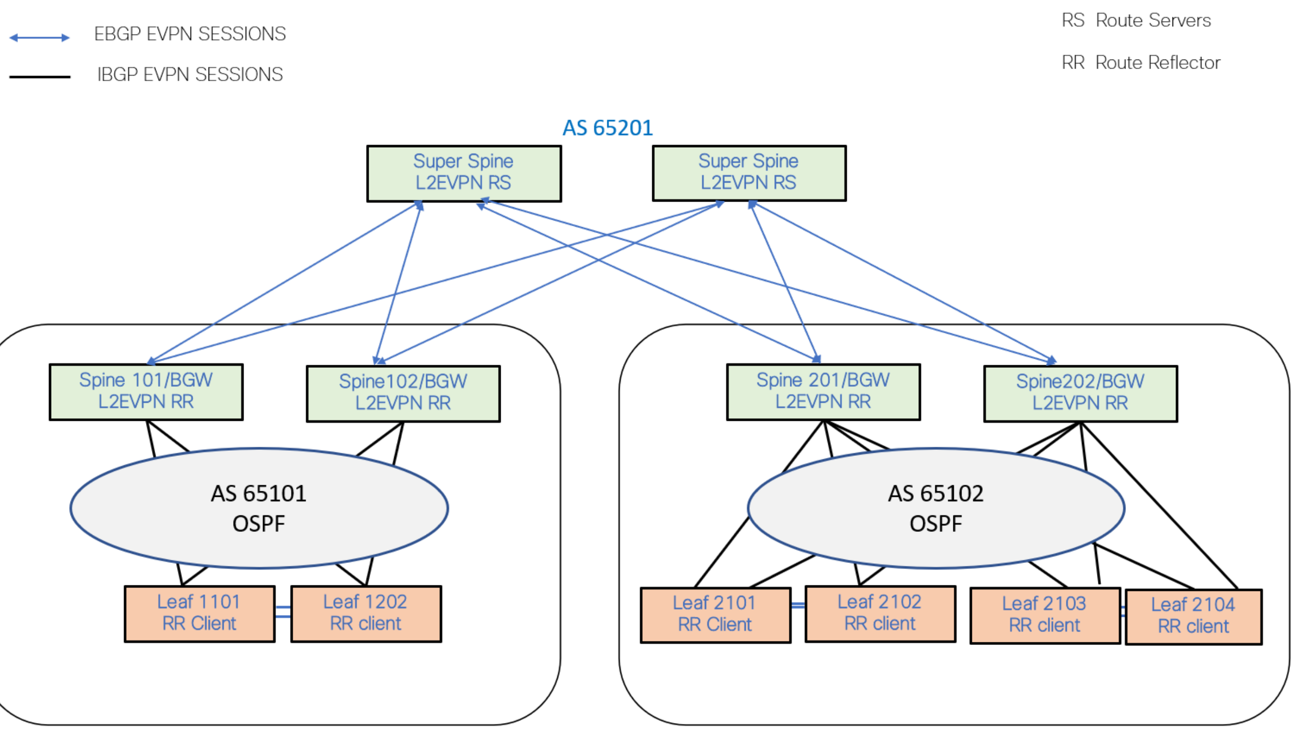


Figure 4. VXLAN Multi-Site Control Plane

Each site is using its own IGP protocol like OSPF or IS-IS for the reachability of VTEPs loopback address and BGP peering loopback address. In this case study OSPF is used as the local site IGP. BGP EVPN is configured between Spine switches/BGW and Leaf switches to exchange EVPN routes. Spine switches will be the Route Reflectors (RR). EVPN routes are exchanged between the spine switches of each site to other sites with next hop (NH) not being changed. The super spine routers or inter site network (ISN) can act as route servers to reflect the EVPN routes of each site to other sites.

VXLAN Multi-Site- Data Plane

While connecting to different VXLAN fabric sites, each site has one or more border gateways (BGW) to other sites via the ISN. BGWs of each site terminate VXLAN tunnels from their own site and reoriginate tunnels from themselves to another site. The VXLAN tunnels are depicted using orange lines in the topology below. The endpoint and routing information between the sites are exchanged between the border gateways of the sites using BGP EVPN control plane.

All the border gateways of each site share the same anycast tunnel address, so that the VXLAN frame from the other site can reach any of the border gateways.

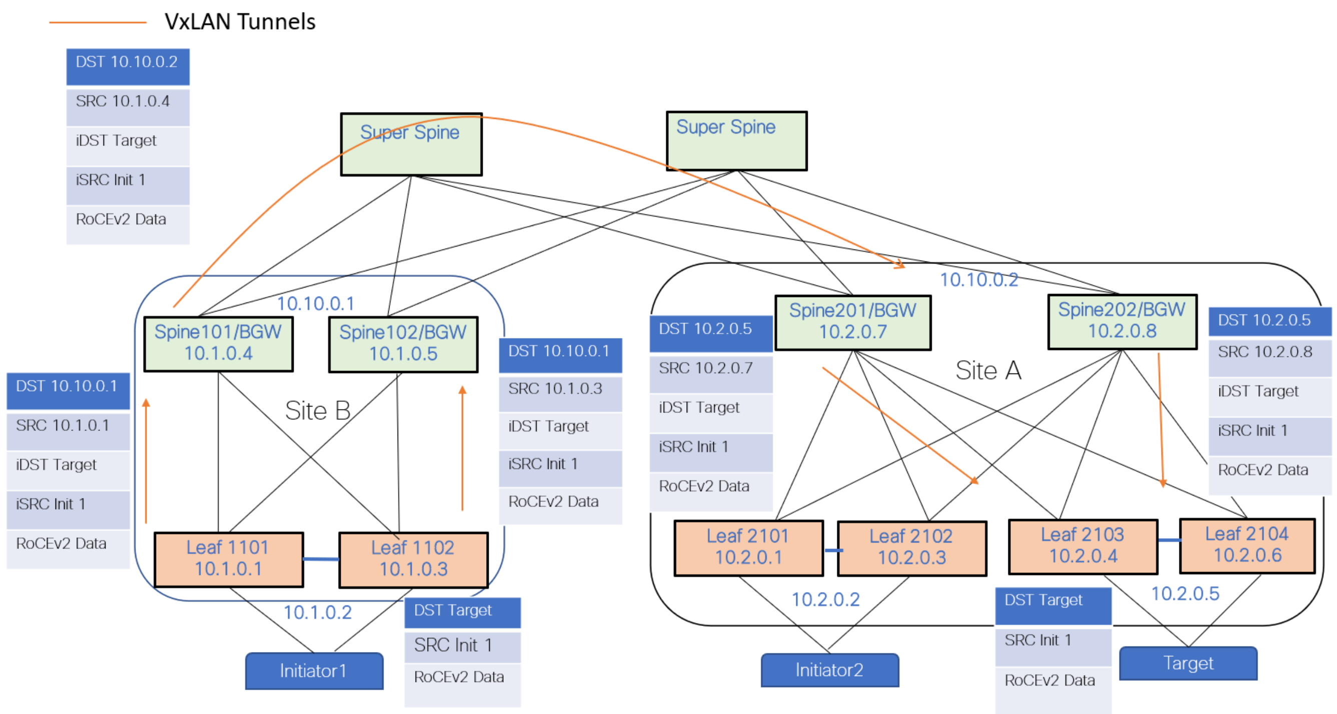


Figure 5. VXLAN Multi-Site Data Plane

Initiators: Servers which initiate read or write operation with a Storage device

Targets: Storage devices.

The above topology explains the data path for the traffic originated from Initiator1 in Site B to Target in Site A.

- RoCEv2 traffic is received on a vpc pair Leaf1101 - Leaf1102 at site B.
- At VPC Pair Leaf1101-Leaf1102, the received traffic is encapsulated on the VXLAN header with source IP of the leaf switch's VTEP address and the destination of the BGW anycast address of its own site.
- At local BGW-siteB, the VXLAN frame is re-originated with the SRC IP of the BGW IP and the destination, the anycast address of the remote site BGWs is Site A in this case.
- At site-A BGW, the VXLAN frame is re-originated with SRC IP of that BGW which receives the frame, and the destination VTEP IP of the VPC Pair where the target is located.

Achieving Lossless Behavior on VXLAN Fabric for RoCEv2 Traffic using PFC

To implement PFC for RoCEv2 traffic on a VXLAN fabric, first, let's try to understand how we can classify the RoCEv2 traffic from other traffic. The best way to classify is by CoS values or by DSCP values rather than port numbers or IP addresses.

In this case study, RoCEv2 data traffic is classified by DSCP 24 and the control packets like CNP are classified with DSCP 48. The below NxOS CLI depicts the classification QoS config.

```
class-map type qos match-any CNP
  match dscp 48
```

```
class-map type qos match-any ROCEv2
  match dscp 24
```

During a read/write operation between an initiator and target, all the traffic is with DSCP value 24. When congestion occurs, PFC frames are generated at the congestion experiencing node towards the traffic source. After congestion is induced, the PFC frames are sent hop-by-hop towards the traffic source. Upon receiving a PFC pause frame, the traffic source reduces the rate at which the traffic is being sent. This process continues till the congestion is fully mitigated.

To implement PFC, priority flow control must be enabled on all the interfaces throughout the network fabric using the CLI “**priority-flow-control mode on**”.

RoCEv2 traffic must be put in the right queue, in this example, RoCEv2 is classified to queue 3 and RoCE control classified to priority queue which is queue 7.

```
policy-map type qos QOS_MARKING
  class ROCEv2
    set qos-group 3
  class CNP
    set qos-group 7
  class class-default
    set qos-group 0


interface Ethernet1/1
  service-policy type qos input QOS_MARKING
  priority-flow-control mode on
```

RoCEv2 classification is required on VTEP interface to classify the decapsulating traffic. VTEP nodes, by default in uniform mode, copies the QoS DSCP values from the inner IP header to the outer VXLAN header during encapsulation and the reverse process during decapsulation of the VXLAN header.

```
interface nve1
  service-policy type qos input QOS_MARKING
```

To generate pause frame on a specific queue, the following network QoS configuration should be configured. In this case study, PFC pause frames are generated on queue 3. MTU must match the RoCEv2 traffic MTU for proper generation of pause frames when congestion occurs.

```
policy-map type network-qos qos_network
  class type network-qos c-8q-nq3
    pause pfc-cos 3
    mtu 9216
  class type network-qos c-8q-nq-default
    mtu 9216
```

The network QoS config and the QoS queueing/scheduling configuration must be applied on the system level to apply for all the interfaces in the system.

```
system qos
  service-policy type queuing output
  QOS_EGRESS_PORT service-policy type network-qos qos_network
```

PFC walk through on VXLAN fabric

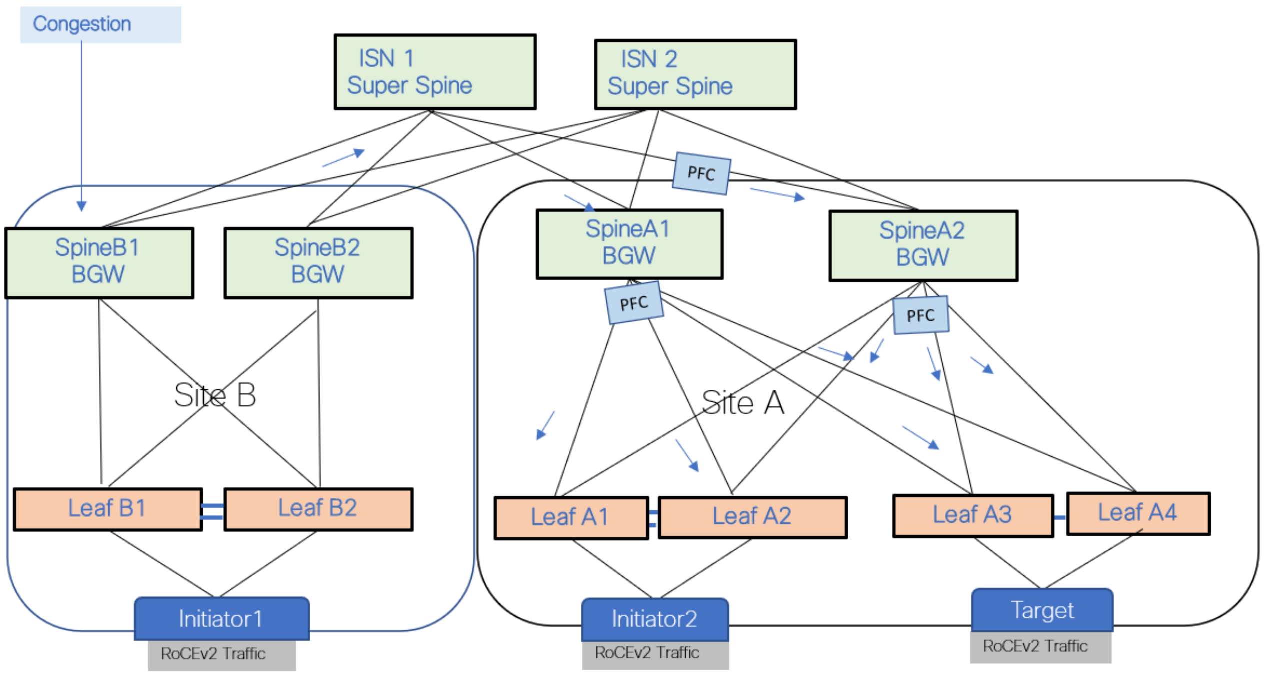


Figure 6. PFC flow on VXLAN Multi-Site Fabric

In the above VXLAN fabric topology there are two sites A & B. There are initiators on Site A and Site B and Target on Site A. Both the initiators are performing a read operation from the target.

PFC is configured on all the leaf nodes, spine/BGW nodes, and ISN so that the PFC hop-by-hop behavior is intact. QOS classification and queuing configuration are also done on all the nodes.

When Spine B1 towards leaf B1 experiences congestion, Spine B1 generates PFC towards ISN which in turn generates PFC towards Spine A1 or A2 hop by hop till it reaches the target to reduce the traffic rate. When there are multiple uplinks contributing to the congestion, PFC is generated towards the uplink which contributes towards the congestion rather than sending PFC on all the uplinks equally.

PFC generation can be verified using the following CLI:

```
show interface priority-flow-control

============================================================
Port            Mode    Oper (VL bmap)    RxPPP      TxPPP
============================================================
Ethernet1/1     On      On   (8)          9638881    12024
Ethernet1/2     On      On   (8)          0          0
Ethernet1/3     On      On   (8)          0          511
Ethernet1/4     On      On   (8)          35079      1205
Ethernet1/5     Auto    Off               0          0
Ethernet1/6     Auto    Off               0          0
--truncated---
```

```
show interface e1/1 priority-flow-control detail slot 1
=======
Ethernet1/1
     Admin Mode: On
     Oper Mode: On VL
     bitmap: (8)
     Total Rx PFC Frames: 9638881 Total
     Tx PFC Frames: 12024
```

```
 ------------------------------------------------------------------------------------------------------
 |Priority0|Priority1 |Priority2 |Priority3  |Priority4 |Priority5 |Priority6|Priority7
 ------------------------------------------------------------------------------------------------------
 Rx |0      |0          |0          |9638881    |0          |0          |0         |0              |
 ------------------------------------------------------------------------------------------------------
 Tx |0      |0          |0          |12024      |0          |0          |0         |0              |
 ------------------------------------------------------------------------------------------------------
```

ECN on VXLAN Fabric

Explicit congestion notification bit on IP header is a 2-bit field which is set on the switch for the traffic experiencing congestion. In this case study, congestion was induced on SpineB1 BGW as shown in Figure-7. This ECN notification is always a forward notification. It is up to the receiving application to inform the sender about the congestion.

Two bits for ECN with the following options:

```
0x00    -    Non       ECN-Capable         Transport                          (Non-ECT)
0x10    -    ECN       Capable             Transport            0              (ECT-0)
0x01    -    ECN       Capable             Transport            1              (ECT-1)
0x11    -    Congestion Encountered (CE)
```

If ECN is enabled on the node, during network congestion, the switches change those two bits from 01 or 10 to 11.

For RoCEv2 traffic on VXLAN fabric, when congestion happens, the ECN bit is set on the outer VXLAN header. This ECN bit is copied to the inner header at the decapsulation VTEP. If the congestion occurs on the ingress leaf, the ECN bit is set on the inner header itself. In this case study, the receiving storage application informs the traffic source with CNP packets to reduce the rate. CNP is a special RoCEv2 packet with source UDP ports set to zero, destination set to 4791, and DSCP value set to 48 on the IP header. VXLAN fabric is configured with QOS classifying CNP packets to a high-priority queue. CNP packets are never dropped in the network because of the high-priority classification.

To enable ECN on VXLAN Fabric, the first step is to classify RoCEv2 traffic to the right queue. In our case study, RoCEv2 traffic is identified with DSCP 24 and classified to queue 3 and the CNP packets are classified to high priority queue.

```
policy-map type qos QOS_MARKING
  class ROCEv2
    set qos-group 3
  class CNP
```

```
    set qos-group 7
  class class-default
    set qos-group 0

interface Ethernet1/1
  service-policy type qos input QOS_MARKING
```

In this case study, ECN is enabled on queue 3 with a minimum threshold of 150 Kbytes and a maximum threshold of 3000 Kbytes.

```
policy-map type queuing
  QOS_EGRESS_PORT class type queuing c-out-8q-q6
    bandwidth remaining percent 0
  class type queuing c-out-8q-q5
    bandwidth remaining percent 0
  class type queuing c-out-8q-q4
    bandwidth remaining percent 0
  class type queuing c-out-8q-q3
    bandwidth remaining percent 80
    random-detect minimum-threshold 150 Kbytes maximum-threshold 3000 kbytes drop-
probability 7 weight 0 ecn
  class type queuing c-out-8q-q2
    bandwidth remaining
    percent 0 class type queuing
    c-out-8q-q1
```

ECN Walk-through over VXLAN Fabric

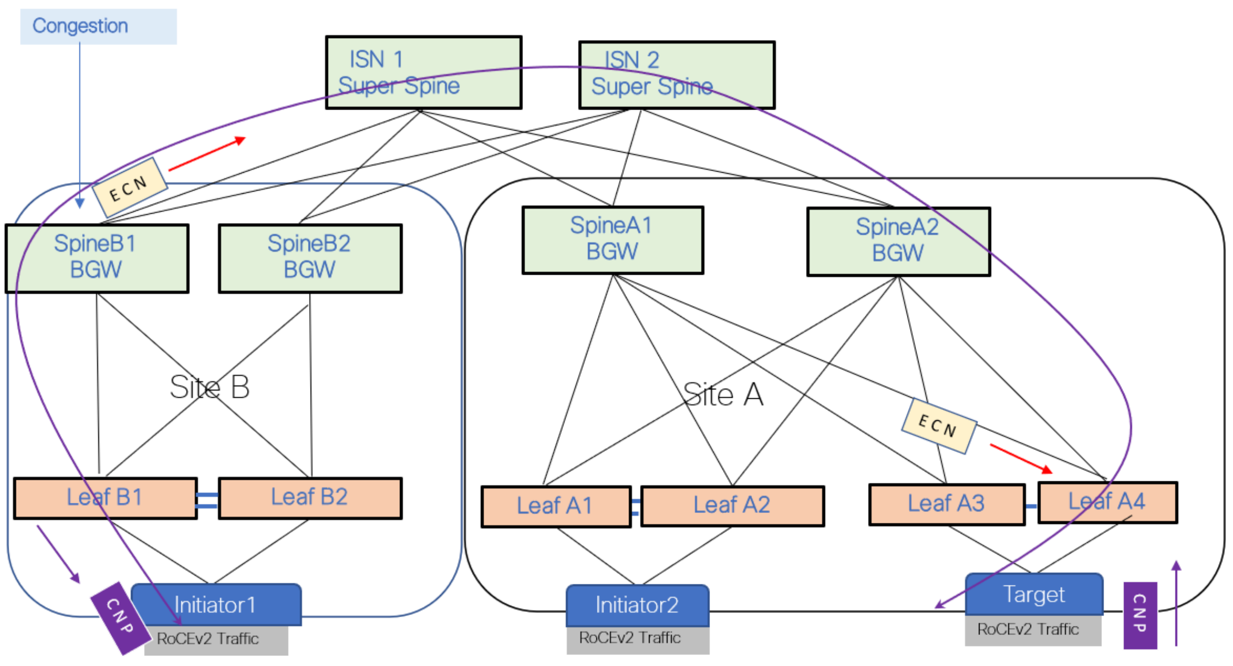


Figure 7. ECN on VXLAN Multi-Site Fabric

On the above VXLAN multi-site fabric, Initiator 1 on Site B and Initiator 2 on Site A are performing a read-and-write operation with the target at Site A. When congestion occurs at Site B, Spine B1/BGW, if the traffic exceeds the WRED threshold on queue 3, ECN bits are set on the outer header of the VXLAN packet at Spine B1/BGW. Spine A1 decapsulates the VXLAN packet and re-originates the packet towards Leaf A3. The ECN bits are copied to the inner header wherever decapsulation happens. After the target receives the RoCEv2 packet with ECN bit set, it will originate a CNP packet to Initiator 1 at Site B. Initiator 1, upon receiving the CNP reduces the read rate from the Target. Read/write operations between Initiator 2 and the target are never disturbed during this operation.

Queuing and Scheduling

The queuing and scheduling process allows you to control the bandwidth allocated to traffic classes so that you achieve the desired trade-off between throughput and latency.

You can apply weighted random early detection (WRED) to a class of traffic, which allows packets to be dropped based on the QoS group. The WRED algorithm allows you to perform proactive queue management to avoid traffic congestion.

You can shape traffic by imposing a maximum data rate on a class of traffic so that excess packets are retained in a queue. You can shape traffic by imposing a maximum data rate on a class of traffic so that excess packets are retained in a queue to shape and regulate the output rate. In addition, minimum bandwidth shaping can be configured to provide a minimum guaranteed bandwidth for a class of traffic. In addition, minimum bandwidth shaping can be configured to provide a minimum guaranteed bandwidth for a class of traffic.

You can limit the size of the queues for a particular class of traffic by applying either static or dynamic limits.

WRED ECN extension allows to mark packets instead of dropping when the average queue length exceeds a specific threshold value.

AFD is flow aware while WRED is not, the recommendation is to test AFD or WRED on their own environment and choose if it meets the dynamic nature of the storage traffic.

Approximate Fair Dropping (AFD) with Elephant Trap

AFD is an active queue-management scheme whose fundamental goal is to provide fair bandwidth allocation among flows that share a common egress queue. Fairness has two aspects. First, AFD distinguishes long-lived elephant flows from short-lived mice flows and exempts mice flows from the dropping algorithm so that mice flows get their fair share of bandwidth without being starved by bandwidth-hungry elephant flows. Second, AFD tracks elephant flows and subjects them to the AFD algorithm in the egress queue to grant them their fair share of bandwidth.

Elephant Trap (ETRAP)

AFD uses ETRAP to distinguish long-lived elephant flows from short-lived mice flows. A flow may be defined using multiple parameters, but typically the 5-tuple is used. ETRAP operates on the ingress side of a switch. It measures the byte counts of incoming flows and compares this against the ETRAP threshold that is defined in bytes. Flows with a byte count lower than the threshold are mice flows. After a flow crosses the threshold, it becomes an elephant flow and is moved to the elephant table for tracking. The

ETRAP threshold is user configurable to allow customer traffic patterns to dictate what constitutes elephant flow.

The elephant table stores and tracks elephant flows. Elephant flows in the table are measured to determine their data arrival rate and their activity. The measured data rates are passed to the buffer management mechanism on egress queues, where the rates are used by the AFD algorithm to calculate the probability of drops for each flow. Elephant flows are aged out if they do not remain active for the configured timeout period. A user-configured, age-period timer, and a bandwidth threshold are used to evaluate the liveliness of an elephant flow. When its average bandwidth during the age-period time is lower than the configured bandwidth threshold, an elephant flow is inactive and will time-out and removed from the elephant flow table

`show hardware flow etrap` shows all the elephant flows that are likely to get the ECN bit set on that specific queue.

```
show hardware flow etrap
Elephant Flows
=====================================================================================
Unit:Slc Index:Type Source Address  Destination Address  Ports(Src:Dst)    Proto Approx_Rate
=====================================================================================
   0:0        0:2 10.108.1.4        17.1.1.4            1024:4791 17         264.320 Mb
   0:0        1:2 10.108.1.8        17.1.1.8            1024:4791 17         264.000 Mb
   0:0        2:2 17.1.1.9          10.108.1.9          1024:4791 17         518.560 Mb
   0:0        4:2 10.106.1.3        18.1.1.3            1024:4791 17         295.840 Mb
```

Differences between WRED and AFD

Although WRED and AFD are both AQM algorithms, they have different approaches to manage congestion:

- WRED computes a random drop probability and drops the packets indiscriminately across all the flows in a class of traffic.
- AFD computes drop probability based on the arrival rate of incoming flows, compares it with the computed fair rate, and drops the packets from the elephant flows with no impact to the mice flows.
- AFD and WRED cannot be applied at the same time. Only one can be used in a system.

ECN can be generated either when traffic exceeds the WRED thresholds or when it exceeds the desired queue threshold of Approximate Fair dropping (AFD). AFD on PFC queue can be clubbed with dynamic packet prioritization (dpp) for mice flows in the default queue. The config example below has the same classification as discussed above but, on the network-qos, we have enabled 'dpp' to match the mice flows on the default queue. This config matches any mice flows like 'ssh' traffic and move those flows to high-priority queue.

On PFC enabled queue, ECN is generated when the AFD thresholds are exceeded.

```
policy-map type network-qos qos_network
  class type network-qos c-8q-nq3
    pause pfc-cos 3
    mtu 9216
  class type network-qos c-8q-nq-default
```

```
    mtu 9216
    dpp set-qos-group 7

policy-map type queuing 25G-QOS_EGRESS_PORT
  class type queuing c-out-8q-q3
    bandwidth remaining percent 80
    afd queue-desired 375 kbytes ecn
  class type queuing c-out-8q-q-default
    bandwidth remaining percent 20
    afd queue-desired 375 kbytes
  class type queuing c-out-8q-q7
    priority level 1
```

Initiator and Target Configurations

Cisco UCS servers running Linux with Mellanox Network adapters are the initiators and targets in the case study. The OFED driver is installed on the server. The OFED (OpenFabrics Enterprise Distribution) is open-source software for RDMA and kernel bypass.

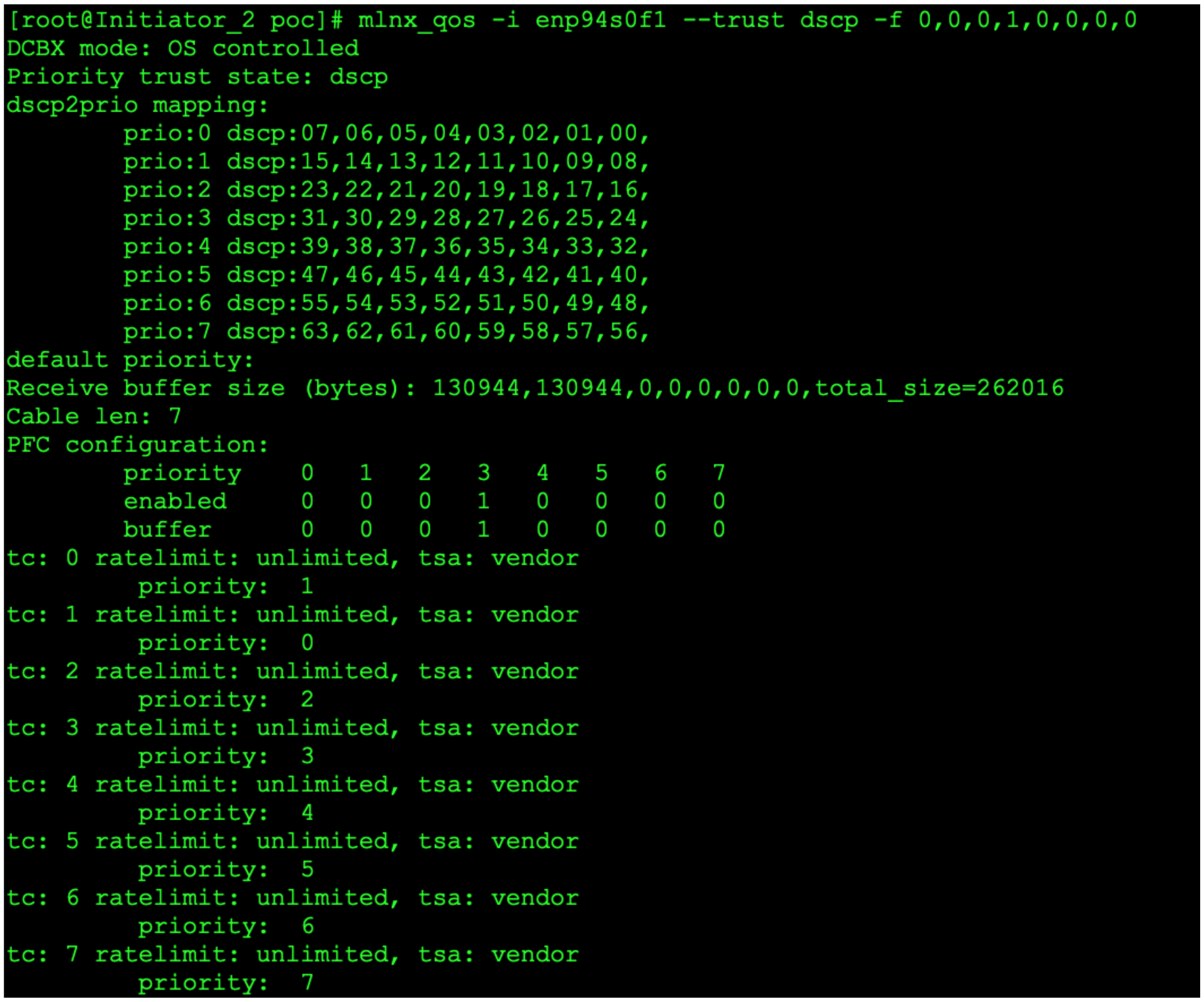
Mellanox OFED (MLNX_OFED) is a Mellanox tested and packaged version of OFED that supports two interconnect types using the same RDMA (remote DMA) and kernel bypass APIs called OFED verbs – InfiniBand and Ethernet.

Mellanox Adapters' Linux Drivers for Ethernet and InfiniBand are available in all the major distributions, the matching Linux Kernel is needed when using inbox OFED driver.

Initiator Configuration

- Configure Driver and Library for NVME:
  - Display the drive detail & modinfo detail -- **show the nvme driver detail**
    ```
    $ ofed_info
    $ modprobe rdma_cm
    $ modprobe nvme-rdma
    $ modprobe mlx5_ib
    $ lsmod | grep nvme
    ```
  - Display the ConnectX-5 Driver setting: -- ECN and CNP setting are correct
    ```
    $ ibdev2netdev
    $ mstconfig -d 5e:00.0 q | grep ECN --> 0
    $ mstconfig -d 5e:00.1 q | grep CNP --> 48
    $ mstconfig -d 5e:00.0 q | grep ECN --> 0
    $ mstconfig -d 5e:00.1 q | grep CNP --> 48
    ```
- Configure DSCP value for the NIC:

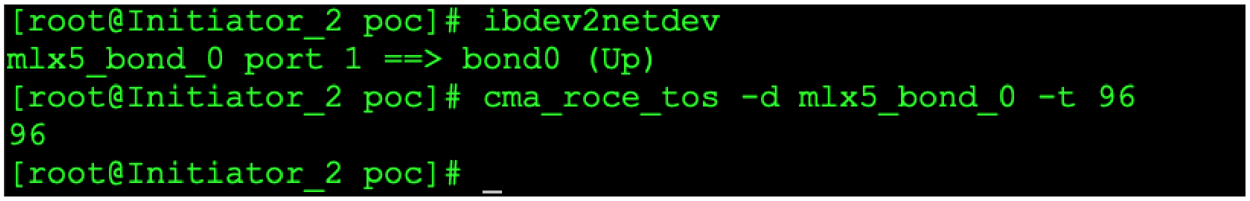
```
$ mlnx_qos -i enp94s0f1 --trust dscp; -f 0,0,0,1,0,0,0,0
```



```
[root@Initiator_2 poc]# mlnx_qos -i enp94s0f1 --trust dscp -f 0,0,0,1,0,0,0,0
DCBX mode: OS controlled
Priority trust state: dscp
dscp2prio mapping:
        prio:0 dscp:07,06,05,04,03,02,01,00,
        prio:1 dscp:15,14,13,12,11,10,09,08,
        prio:2 dscp:23,22,21,20,19,18,17,16,
        prio:3 dscp:31,30,29,28,27,26,25,24,
        prio:4 dscp:39,38,37,36,35,34,33,32,
        prio:5 dscp:47,46,45,44,43,42,41,40,
        prio:6 dscp:55,54,53,52,51,50,49,48,
        prio:7 dscp:63,62,61,60,59,58,57,56,
default priority:
Receive buffer size (bytes): 130944,130944,0,0,0,0,0,0,total_size=262016
Cable len: 7
PFC configuration:
        priority    0   1   2   3   4   5   6   7
        enabled     0   0   0   1   0   0   0   0
        buffer      0   0   0   1   0   0   0   0
tc: 0 ratelimit: unlimited, tsa: vendor
         priority:  1
tc: 1 ratelimit: unlimited, tsa: vendor
         priority:  0
tc: 2 ratelimit: unlimited, tsa: vendor
         priority:  2
tc: 3 ratelimit: unlimited, tsa: vendor
         priority:  3
tc: 4 ratelimit: unlimited, tsa: vendor
         priority:  4
tc: 5 ratelimit: unlimited, tsa: vendor
         priority:  5
tc: 6 ratelimit: unlimited, tsa: vendor
         priority:  6
tc: 7 ratelimit: unlimited, tsa: vendor
         priority:  7
```

- Configure DSCP value for RoCE Traffic

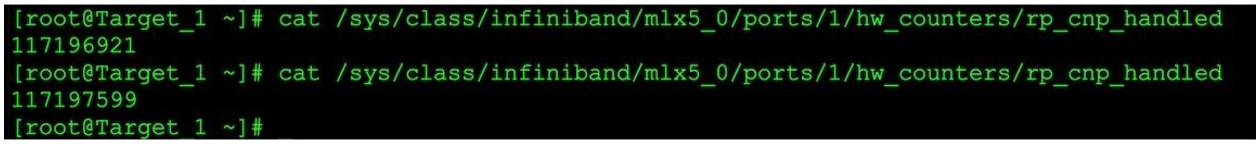
```
$ cma_roce_tos -d mlx5_bond_0 -t 96
```



```
[root@Initiator_2 poc]# ibdev2netdev
mlx5_bond_0 port 1 ==> bond0 (Up)
[root@Initiator_2 poc]# cma_roce_tos -d mlx5_bond_0 -t 96
96
[root@Initiator_2 poc]# _
```

- Verify the interface CNP counter:

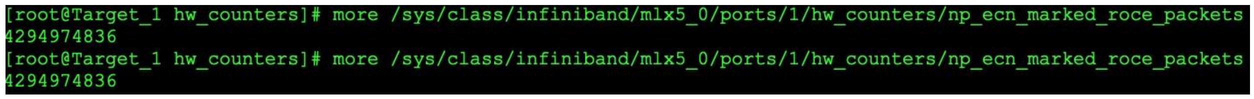
```
$ cat /sys/class/infiniband/mlx5_0/ports/1/hw_counters/np_cnp_sent
```



```
[root@Target_1 ~]# cat /sys/class/infiniband/mlx5_0/ports/1/hw_counters/rp_cnp_handled
117196921
[root@Target_1 ~]# cat /sys/class/infiniband/mlx5_0/ports/1/hw_counters/rp_cnp_handled
117197599
[root@Target_1 ~]#
```

- Verify the interface ECN counter:

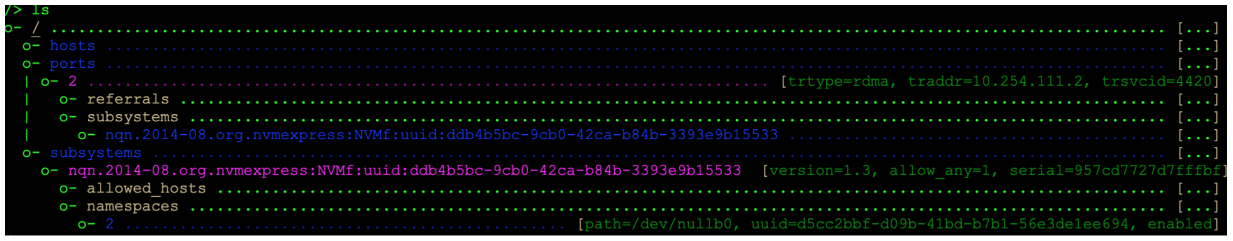
```
$ /sys/class/infiniband/mlx5_0/ports/1/hw_counters/np_ecn_marked_roce_packets
```



```
[root@Target_1 hw_counters]# more /sys/class/infiniband/mlx5_0/ports/1/hw_counters/np_ecn_marked_roce_packets
4294974836
[root@Target_1 hw_counters]# more /sys/class/infiniband/mlx5_0/ports/1/hw_counters/np_ecn_marked_roce_packets
4294974836
```

Target Configuration 1: Use UCS Server as Target

The UCS server can be converted into target. It is an easy way to run benchmark storage traffic test between servers. Use the **nvmetcli** utility to edit, view, and start an NVMe target on the UCS server. Use a null block device to assign to the nvmet subsystem. This block is used to communicate with the Initiators. Here is snapshot of NVMe Target on UCS server:

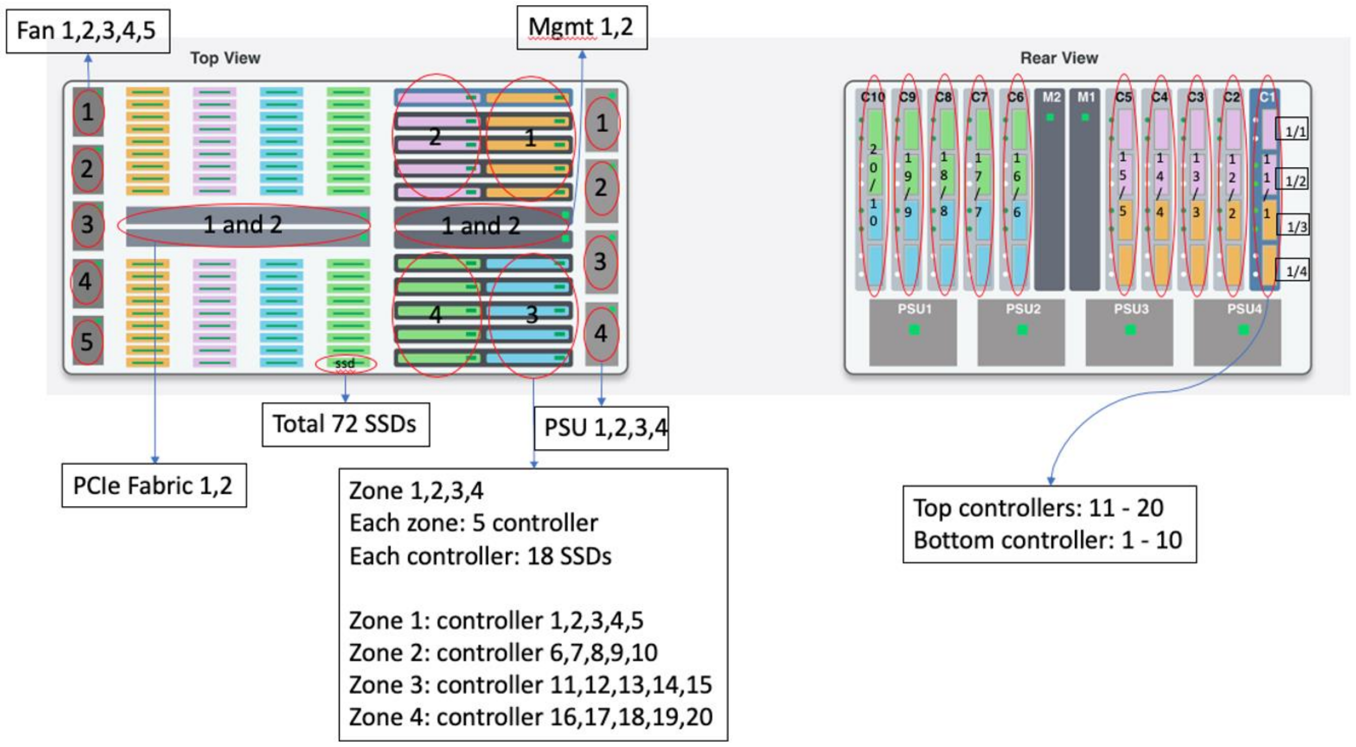


```
/> ls
o- / ......................................................................................... [...]
  o- hosts ................................................................................... [...]
  o- ports ................................................................................... [...]
  | o- 2 ................................................... [trtype=rdma, traddr=10.254.111.2, trsvcid=4420]
  |   o- referrals ........................................................................... [...]
  |   o- subsystems .......................................................................... [...]
  |     o- nqn.2014-08.org.nvmexpress:NVMf:uuid:ddb4b5bc-9cb0-42ca-b84b-3393e9b15533 ............. [...]
  o- subsystems .............................................................................. [...]
    o- nqn.2014-08.org.nvmexpress:NVMf:uuid:ddb4b5bc-9cb0-42ca-b84b-3393e9b15533  [version=1.3, allow_any=1, serial=957cd7727d7fffbf]
      o- allowed_hosts ....................................................................... [...]
      o- namespaces .......................................................................... [...]
        o- 2 ................................ [path=/dev/nullb0, uuid=d5cc2bbf-d09b-41bd-b7b1-56e3de1ee694, enabled]
```

Target Configuration 2: Pavilion Hyper Flash Array Configuration

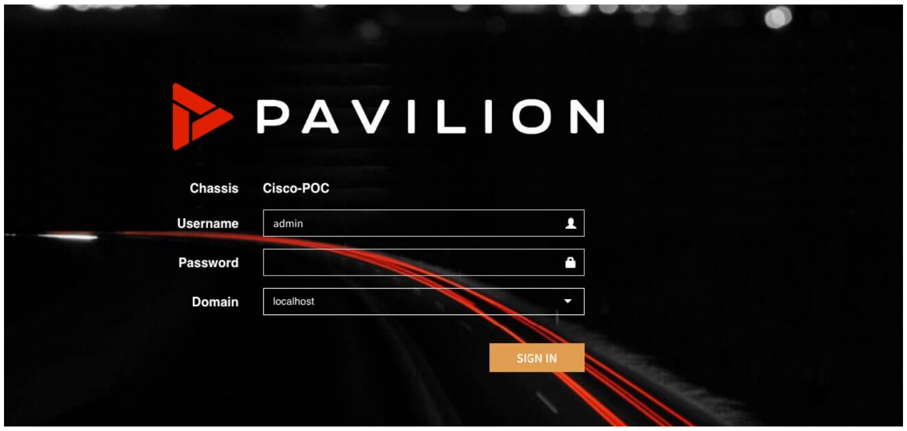
Pavilion Architecture

The pavilion storage platform being used in the lab is the industry's first standard-based network storage device. It consists of 72 PCIe SSDs (Samsung SSD in this case), 20 controllers, 2 PCIe fabric (Gen-3), 2 management modules, 4 power supply unit and 5 Fans. The top and rear view of the device are shown below.



Configuration

Access the pavilion GUI by any kind of browser:



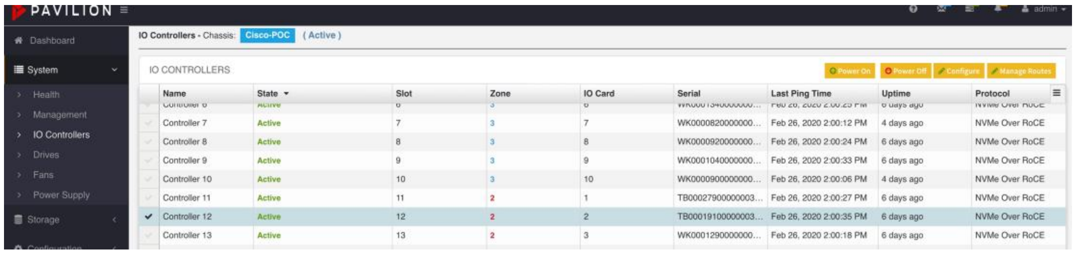
Chassis Cisco-POC
Username admin
Password
Domain localhost

For the detailed configuration guide, see the pavilion's [Pavilion Array GUI Reference Guide](). In this document, we list key components' configurations with our real parameter values in the lab.

How to Configure a Controller?

To Configure a Protocol

Go to system/IO Controllers, select any controller that you like to configure, click '**Configure**' button on the right top corner:

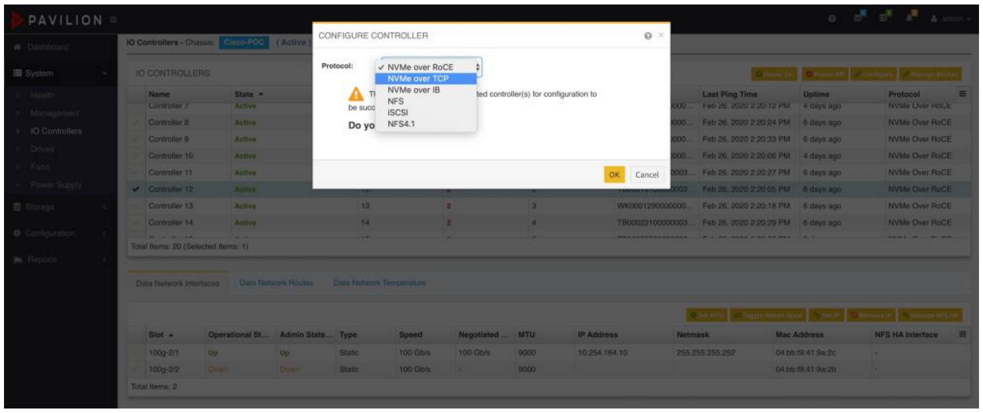


IO Controllers - Chassis: Cisco-POC (Active)

IO CONTROLLERS

	Name	State	Slot	Zone	IO Card	Serial	Last Ping Time	Uptime	Protocol
	Controller 7	Active	7	3	7	WK0000820000000...	Feb 26, 2020 2:00:12 PM	4 days ago	NVMe Over RoCE
	Controller 8	Active	8	3	8	WK0000920000000...	Feb 26, 2020 2:00:24 PM	6 days ago	NVMe Over RoCE
	Controller 9	Active	9	3	9	WK0001040000000...	Feb 26, 2020 2:00:33 PM	6 days ago	NVMe Over RoCE
	Controller 10	Active	10	3	10	WK0000900000000...	Feb 26, 2020 2:00:06 PM	4 days ago	NVMe Over RoCE
	Controller 11	Active	11	2	1	TB00027900000003...	Feb 26, 2020 2:00:27 PM	6 days ago	NVMe Over RoCE
✓	Controller 12	Active	12	2	2	TB00019100000003...	Feb 26, 2020 2:00:35 PM	6 days ago	NVMe Over RoCE
	Controller 13	Active	13	2	3	WK0001290000000...	Feb 26, 2020 2:00:18 PM	6 days ago	NVMe Over RoCE

In the new dialog box, you can see protocols available, such as NVMe over RoCE (version2), NVMe over TCP, NVMe over IB, NFS, and iSCSI and so on. After the protocol is selected, click "**OK**" to complete the controller configuration:



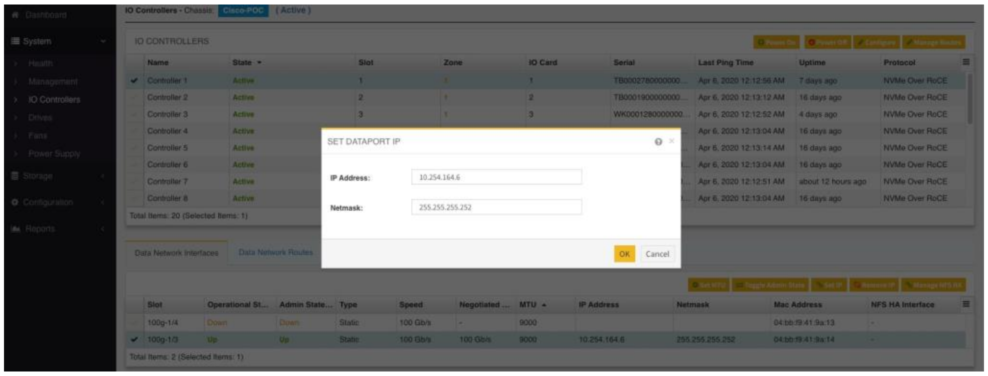
CONFIGURE CONTROLLER

Protocol: ✓ NVMe over RoCE

- NVMe over TCP
- NVMe over IB
- NFS
- iSCSI
- NFS4.1

To Configure an IP Address

After completing the protocol on a controller, you can configure an IP address for a controller through the "**Data Network Interface**" tab on the bottom window; to click "**Set IP**" button at the top-right corner, then assign the IP and Netmask address on the dialog box; to click "**OK**" when it's done.



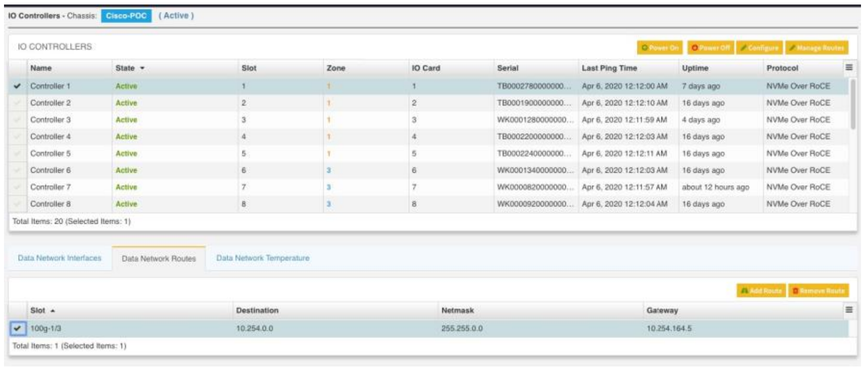
SET DATAPORT IP

IP Address: 10.254.164.6

Netmask: 255.255.255.252

To Configure a Network Route

To move the next tab "**Data Network Routes**", the network routes can be added to a specific controller. To click "**Add Route**" button at the top-right corner, you can add "**destination route**", "**netmask**" and "**gateway**" on the dialog box, then click "**OK**" when it's done.



IO Controllers - Chassis: Cisco-POC (Active)

IO CONTROLLERS

	Name	State	Slot	Zone	IO Card	Serial	Last Ping Time	Uptime	Protocol
✓	Controller 1	Active	1	1	1	TB0002780000000...	Apr 6, 2020 12:12:00 AM	7 days ago	NVMe Over RoCE
	Controller 2	Active	2	1	2	TB0001900000000...	Apr 6, 2020 12:12:10 AM	16 days ago	NVMe Over RoCE
	Controller 3	Active	3	1	3	WK0001280000000...	Apr 6, 2020 12:11:59 AM	4 days ago	NVMe Over RoCE
	Controller 4	Active	4	1	4	TB0002200000000...	Apr 6, 2020 12:12:03 AM	16 days ago	NVMe Over RoCE
	Controller 5	Active	5	1	5	TB0002240000000...	Apr 6, 2020 12:12:11 AM	16 days ago	NVMe Over RoCE
	Controller 6	Active	6	3	6	WK0001340000000...	Apr 6, 2020 12:12:03 AM	16 days ago	NVMe Over RoCE
	Controller 7	Active	7	3	7	WK0000820000000...	Apr 6, 2020 12:11:57 AM	about 12 hours ago	NVMe Over RoCE
	Controller 8	Active	8	3	8	WK0000920000000...	Apr 6, 2020 12:12:04 AM	16 days ago	NVMe Over RoCE

Total Items: 20 (Selected Items: 1)

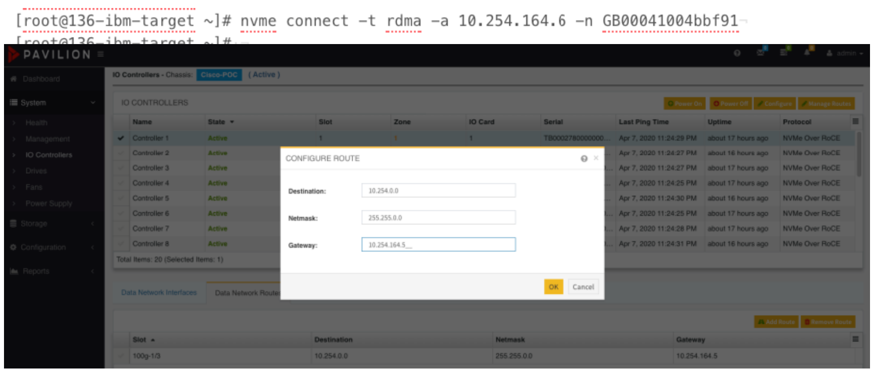
	Slot	Destination	Netmask	Gateway
✓	100g-1/3	10.254.0.0	255.255.0.0	10.254.164.5

Total Items: 1 (Selected Items: 1)

How to Check Initiators?

Frequently, we like to know which initiator is connecting the target (pavilion I/O port on the controller). There is no way to tell by the output of “nvme list” on the initiator.

Pavilion GUI provides a way to accomplish this. Go to “Storage/Initiator” and the output is showing the alive NVMe connections, including initiators’ IP, controller and its I/O number, zone number and so on.



```
[root@136-ibm-target ~]# nvme connect -t rdma -a 10.254.164.6 -n GB00041004bbf91
[root@136-ibm-target ~]#
```

IO Controllers - Chassis: Cisco-POC (Active)

IO CONTROLLERS

Name	State	Slot	Zone	IO Card	Serial	Last Ping Time	Uptime	Protocol
Controller 1	Active	1	1	1	TB00027800000000...	Apr 7, 2020 11:24:29 PM	about 17 hours ago	NVMe Over RoCE
Controller 2	Active					Apr 7, 2020 11:24:27 PM	about 16 hours ago	NVMe Over RoCE
Controller 3	Active					Apr 7, 2020 11:24:27 PM	about 17 hours ago	NVMe Over RoCE
Controller 4	Active					Apr 7, 2020 11:24:25 PM	about 17 hours ago	NVMe Over RoCE
Controller 5	Active					Apr 7, 2020 11:24:30 PM	about 16 hours ago	NVMe Over RoCE
Controller 6	Active					Apr 7, 2020 11:24:25 PM	about 17 hours ago	NVMe Over RoCE
Controller 7	Active					Apr 7, 2020 11:24:28 PM	about 17 hours ago	NVMe Over RoCE
Controller 8	Active					Apr 7, 2020 11:24:31 PM	about 16 hours ago	NVMe Over RoCE

Total Items: 20 (Selected Items: 1)

CONFIGURE ROUTE

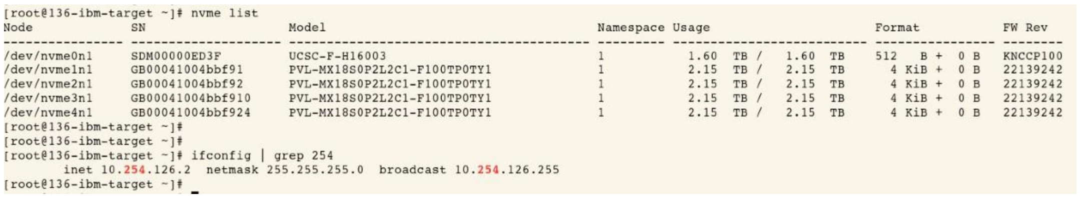
Destination: 10.254.0.0

Netmask: 255.255.0.0

Gateway: 10.254.164.5

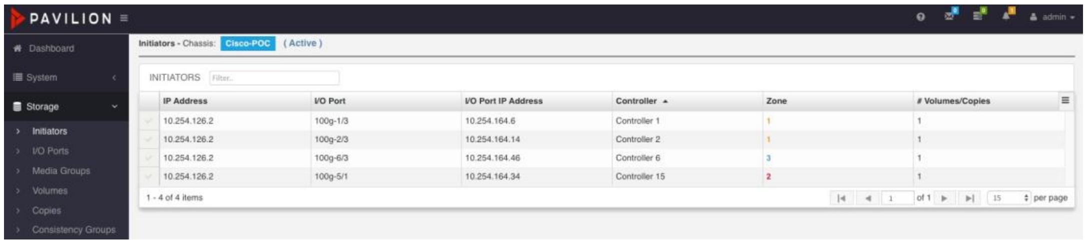
Slot	Destination	Netmask	Gateway
100g-1/3	10.254.0.0	255.255.0.0	10.254.164.5

The above output is the commands for NVMe RoCE connect on the Linux initiator; if for NVMe tcp connect, “**-t tcp -s 4420**” parameters are needed.



```
[root@136-ibm-target ~]# nvme list
Node             SN                   Model                                    Namespace Usage                      Format           FW Rev
---------------- -------------------- ---------------------------------------- --------- -------------------------- ---------------- --------
/dev/nvme0n1     SDM00000ED3F         UCSC-F-H16003                            1           1.60  TB /   1.60  TB    512   B +  0 B   KNCCP100
/dev/nvme1n1     GB00041004bbf91      PVL-MX18S0P2L2C1-F100TP0TY1              1           2.15  TB /   2.15  TB      4 KiB +  0 B   22139242
/dev/nvme2n1     GB00041004bbf92      PVL-MX18S0P2L2C1-F100TP0TY1              1           2.15  TB /   2.15  TB      4 KiB +  0 B   22139242
/dev/nvme3n1     GB00041004bbf910     PVL-MX18S0P2L2C1-F100TP0TY1              1           2.15  TB /   2.15  TB      4 KiB +  0 B   22139242
/dev/nvme4n1     GB00041004bbf924     PVL-MX18S0P2L2C1-F100TP0TY1              1           2.15  TB /   2.15  TB      4 KiB +  0 B   22139242
[root@136-ibm-target ~]#
[root@136-ibm-target ~]#
[root@136-ibm-target ~]# ifconfig | grep 254
        inet 10.254.126.2  netmask 255.255.255.0  broadcast 10.254.126.255
[root@136-ibm-target ~]# _
```

The above output is the result of NVMe RoCE connection on the Linux initiator; “10.254.126.2” is the initiator’s IP address.



Initiators - Chassis: Cisco-POC (Active)

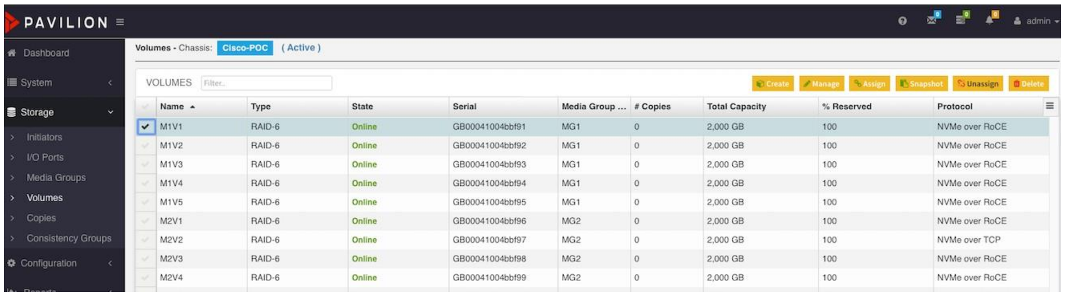
INITIATORS

IP Address	I/O Port	I/O Port IP Address	Controller	Zone	# Volumes/Copies
10.254.126.2	100g-1/3	10.254.164.6	Controller 1	1	1
10.254.126.2	100g-2/3	10.254.164.14	Controller 2	1	1
10.254.126.2	100g-6/3	10.254.164.46	Controller 6	3	1
10.254.126.2	100g-5/1	10.254.164.34	Controller 15	2	1

1 - 4 of 4 items

How to Modify the Protocol from NVMe over ROCE to NVMe over TCP for a volume?

Step 1. To go to “**Storage/Volumes**”, select a volume and then click “**unassign**” in the top-right corner.

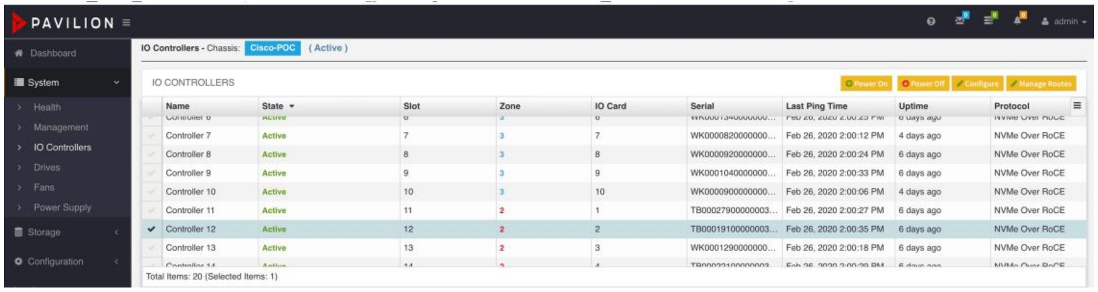


Volumes - Chassis: Cisco-POC (Active)

VOLUMES

Name	Type	State	Serial	Media Group ...	# Copies	Total Capacity	% Reserved	Protocol
M1V1	RAID-6	Online	GB00041004bbf91	MG1	0	2,000 GB	100	NVMe over RoCE
M1V2	RAID-6	Online	GB00041004bbf92	MG1	0	2,000 GB	100	NVMe over RoCE
M1V3	RAID-6	Online	GB00041004bbf93	MG1	0	2,000 GB	100	NVMe over RoCE
M1V4	RAID-6	Online	GB00041004bbf94	MG1	0	2,000 GB	100	NVMe over RoCE
M1V5	RAID-6	Online	GB00041004bbf95	MG1	0	2,000 GB	100	NVMe over RoCE
M2V1	RAID-6	Online	GB00041004bbf96	MG2	0	2,000 GB	100	NVMe over RoCE
M2V2	RAID-6	Online	GB00041004bbf97	MG2	0	2,000 GB	100	NVMe over TCP
M2V3	RAID-6	Online	GB00041004bbf98	MG2	0	2,000 GB	100	NVMe over RoCE
M2V4	RAID-6	Online	GB00041004bbf99	MG2	0	2,000 GB	100	NVMe over RoCE

Step 2. To go to "**System/IO Controller**", select the controller which is associated with the specific volume: then click "**configure**" button on the top-right corner and select a protocol that you like to configure on the dialog box:



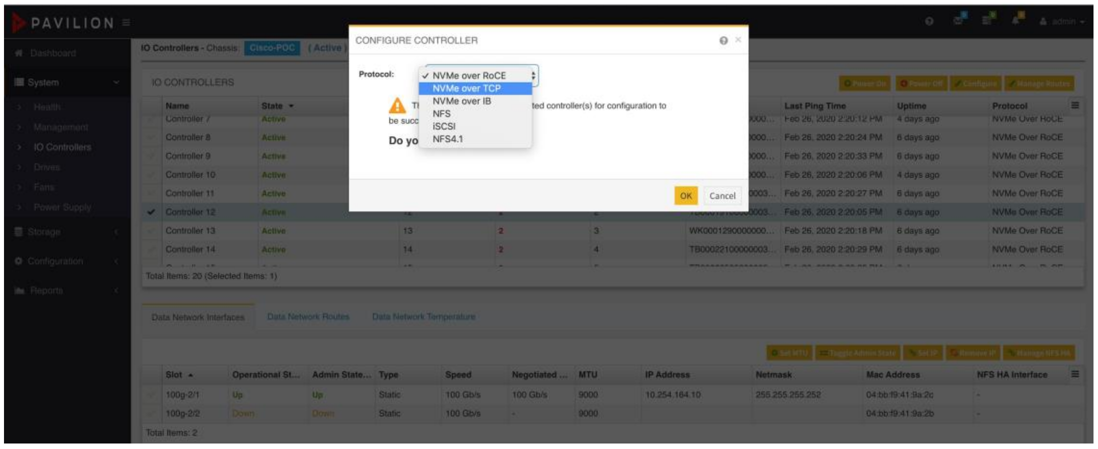
IO Controllers - Chassis: Cisco-POC (Active)

IO CONTROLLERS

Name	State	Slot	Zone	IO Card	Serial	Last Ping Time	Uptime	Protocol
Controller 7	Active	7	3	7	WK0000820000000...	Feb 26, 2020 2:00:12 PM	4 days ago	NVMe Over RoCE
Controller 8	Active	8	3	8	WK0000920000000...	Feb 26, 2020 2:00:24 PM	6 days ago	NVMe Over RoCE
Controller 9	Active	9	3	9	WK0001040000000...	Feb 26, 2020 2:00:33 PM	6 days ago	NVMe Over RoCE
Controller 10	Active	10	3	10	WK0000900000000...	Feb 26, 2020 2:00:06 PM	4 days ago	NVMe Over RoCE
Controller 11	Active	11	2	1	TB00027900000003...	Feb 26, 2020 2:00:27 PM	6 days ago	NVMe Over RoCE
Controller 12	Active	12	2	2	TB00019100000003...	Feb 26, 2020 2:00:35 PM	6 days ago	NVMe Over RoCE
Controller 13	Active	13	2	3	WK0001290000000...	Feb 26, 2020 2:00:18 PM	6 days ago	NVMe Over RoCE

Total Items: 20 (Selected Items: 1)

Step 3. Click "**configure**" button on the top-right corner and select a protocol that you like to configure on the dialog box:



CONFIGURE CONTROLLER

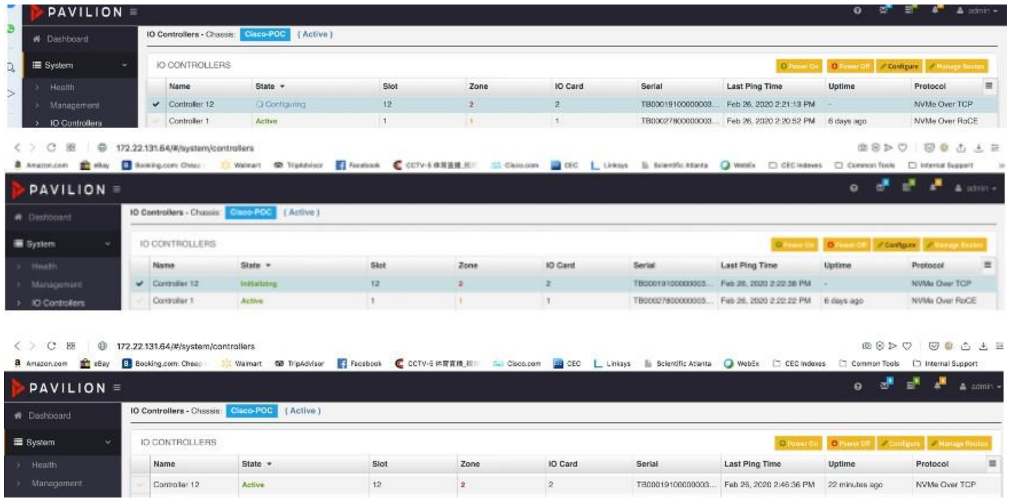
Protocol: ✓ NVMe over RoCE / NVMe over TCP / NVMe over IB / NFS / iSCSI / NFS4.1

Data Network Interfaces

Slot	Operational St...	Admin State...	Type	Speed	Negotiated ...	MTU	IP Address	Netmask	Mac Address	NFS HA Interface
100g-2/1	Up	Up	Static	100 Gb/s	100 Gb/s	9000	10.254.184.10	255.255.255.252	04:bb:f9:41:9a:2c	-
100g-2/2	Down	Down	Static	100 Gb/s	-	9000			04:bb:f9:41:9a:2b	-

Total Items: 2

Step 4. After completing the protocol selection, you can see different transitional state that is shown on the state column, "**Configuring -> Initializing -> active**":



172.22.131.64/#/system/controllers

IO Controllers - Chassis: Cisco-POC (Active)

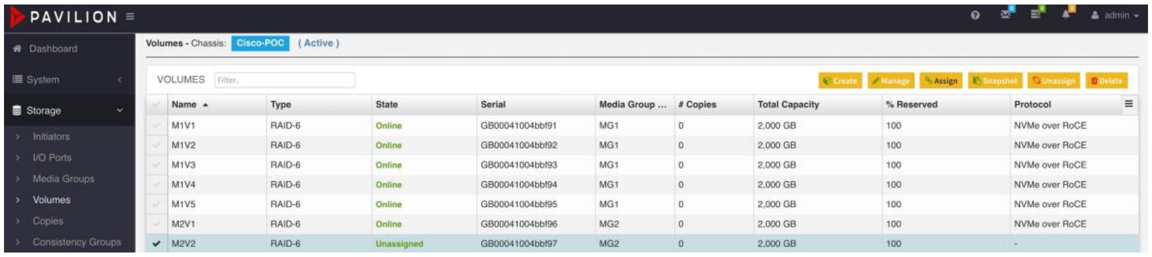
IO CONTROLLERS

	Name	State	Slot	Zone	IO Card	Serial	Last Ping Time	Uptime	Protocol
✓	Controller 12	Configuring	12	2	2	TB00019100000003...	Feb 26, 2020 2:21:13 PM	-	NVMe Over TCP
	Controller 1	Active	1	1	1	TB00027800000003...	Feb 26, 2020 2:20:52 PM	6 days ago	NVMe Over RoCE

	Name	State	Slot	Zone	IO Card	Serial	Last Ping Time	Uptime	Protocol
✓	Controller 12	Initializing	12	2	2	TB00019100000003...	Feb 26, 2020 2:22:38 PM	-	NVMe Over TCP
	Controller 1	Active	1	1	1	TB00027800000003...	Feb 26, 2020 2:22:22 PM	6 days ago	NVMe Over RoCE

	Name	State	Slot	Zone	IO Card	Serial	Last Ping Time	Uptime	Protocol
	Controller 12	Active	12	2	2	TB00019100000003...	Feb 26, 2020 2:46:36 PM	22 minutes ago	NVMe Over TCP

Step 5. To go back to "**Storage/Volumes**", select the original volume with the present "unassigned" state, then click "**assign**" button on the top-right corner.

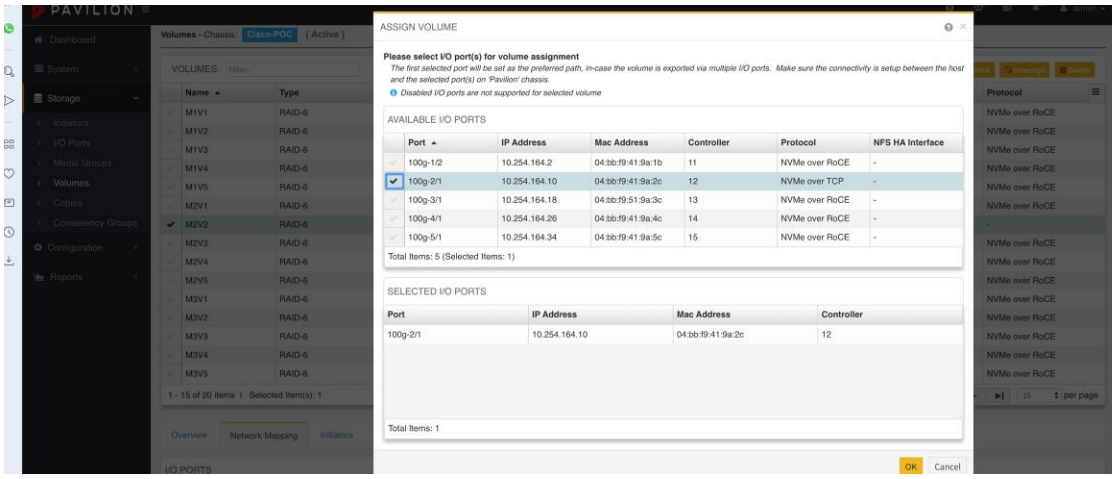


Volumes - Chassis: Cisco-POC (Active)

VOLUMES

	Name	Type	State	Serial	Media Group ...	# Copies	Total Capacity	% Reserved	Protocol
	M1V1	RAID-6	Online	GB00041004bbf91	MG1	0	2,000 GB	100	NVMe over RoCE
	M1V2	RAID-6	Online	GB00041004bbf92	MG1	0	2,000 GB	100	NVMe over RoCE
	M1V3	RAID-6	Online	GB00041004bbf93	MG1	0	2,000 GB	100	NVMe over RoCE
	M1V4	RAID-6	Online	GB00041004bbf94	MG1	0	2,000 GB	100	NVMe over RoCE
	M1V5	RAID-6	Online	GB00041004bbf95	MG1	0	2,000 GB	100	NVMe over RoCE
	M2V1	RAID-6	Online	GB00041004bbf96	MG2	0	2,000 GB	100	NVMe over RoCE
✓	M2V2	RAID-6	Unassigned	GB00041004bbf97	MG2	0	2,000 GB	100	-

Step 6. After clicking "**assign**" button, to select the port which maps to the volume on the dialog box, click "**OK**" when it's done.



ASSIGN VOLUME

Please select I/O port(s) for volume assignment

The first selected port will be set as the preferred path, in-case the volume is exported via multiple I/O ports. Make sure the connectivity is setup between the host and the selected port(s) on 'Pavilion' chassis.

Disabled I/O ports are not supported for selected volume

AVAILABLE I/O PORTS

	Port	IP Address	Mac Address	Controller	Protocol	NFS HA Interface
	100g-1/2	10.254.164.2	04:bb:f9:41:9a:1b	11	NVMe over RoCE	-
✓	100g-2/1	10.254.164.10	04:bb:f9:41:9a:2c	12	NVMe over TCP	-
	100g-3/1	10.254.164.18	04:bb:f9:51:9a:3c	13	NVMe over RoCE	-
	100g-4/1	10.254.164.26	04:bb:f9:41:9a:4c	14	NVMe over RoCE	-
	100g-5/1	10.254.164.34	04:bb:f9:41:9a:5c	15	NVMe over RoCE	-

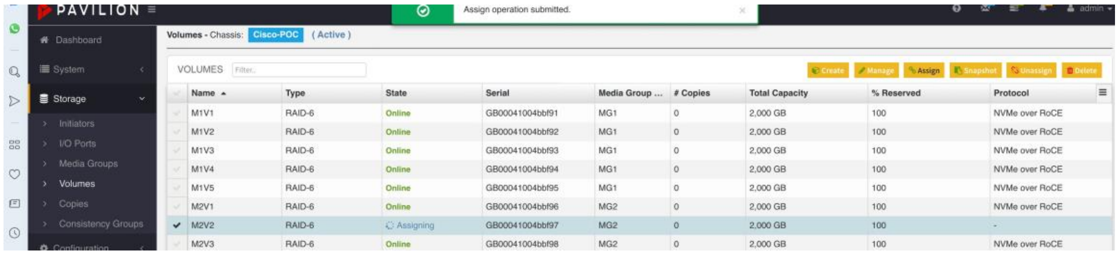
Total Items: 5 (Selected Items: 1)

SELECTED I/O PORTS

Port	IP Address	Mac Address	Controller
100g-2/1	10.254.164.10	04:bb:f9:41:9a:2c	12

Total Items: 1

Step 7. After completing the volume assignment, you can see different transient states that are shown on the state column, "**assigning -> active**"



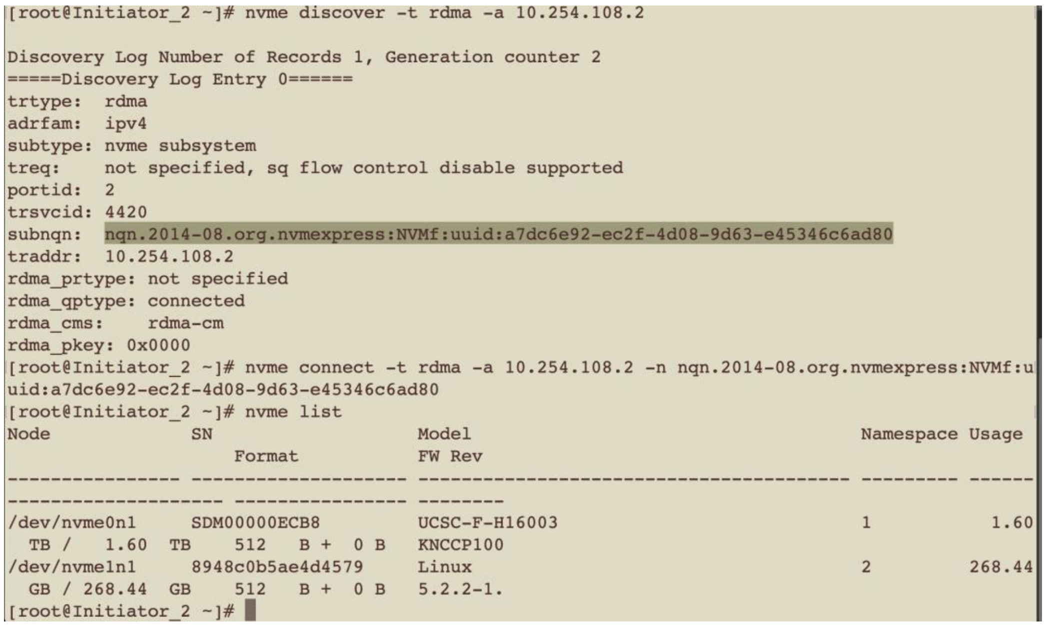
	Name	Type	State	Serial	Media Group ...	# Copies	Total Capacity	% Reserved	Protocol
	M1V1	RAID-6	Online	GB00041004bbf91	MG1	0	2,000 GB	100	NVMe over RoCE
	M1V2	RAID-6	Online	GB00041004bbf92	MG1	0	2,000 GB	100	NVMe over RoCE
	M1V3	RAID-6	Online	GB00041004bbf93	MG1	0	2,000 GB	100	NVMe over RoCE
	M1V4	RAID-6	Online	GB00041004bbf94	MG1	0	2,000 GB	100	NVMe over RoCE
	M1V5	RAID-6	Online	GB00041004bbf95	MG1	0	2,000 GB	100	NVMe over RoCE
	M2V1	RAID-6	Online	GB00041004bbf96	MG2	0	2,000 GB	100	NVMe over RoCE
✓	M2V2	RAID-6	Assigning	GB00041004bbf97	MG2	0	2,000 GB	100	-
	M2V3	RAID-6	Online	GB00041004bbf98	MG2	0	2,000 GB	100	NVMe over RoCE

Establishing NVMe Connection:

Here is an example shows basic NVME operation on Linux: Discover, Connect, and Disconnect. The remote target: 10.254.108.2.

- NVME Discover:nvme discover **-t rdma -a 10.254.108.2**

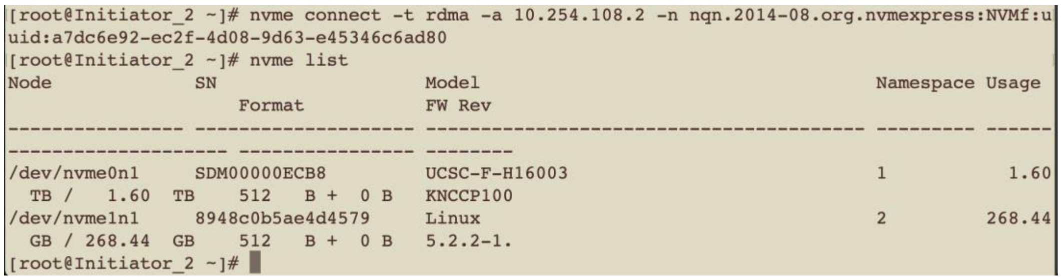
  The output contains the 'subnqn' value for the remote NVME target



```
[root@Initiator_2 ~]# nvme discover -t rdma -a 10.254.108.2

Discovery Log Number of Records 1, Generation counter 2
=====Discovery Log Entry 0======
trtype:  rdma
adrfam:  ipv4
subtype: nvme subsystem
treq:    not specified, sq flow control disable supported
portid:  2
trsvcid: 4420
subnqn:  nqn.2014-08.org.nvmexpress:NVMf:uuid:a7dc6e92-ec2f-4d08-9d63-e45346c6ad80
traddr:  10.254.108.2
rdma_prtype: not specified
rdma_qptype: connected
rdma_cms:    rdma-cm
rdma_pkey: 0x0000
[root@Initiator_2 ~]# nvme connect -t rdma -a 10.254.108.2 -n nqn.2014-08.org.nvmexpress:NVMf:u
uid:a7dc6e92-ec2f-4d08-9d63-e45346c6ad80
[root@Initiator_2 ~]# nvme list
Node             SN                   Model                                    Namespace Usage
                     Format           FW Rev
---------------- -------------------- ---------------------------------------- --------- ------
-------------------- ---------------- --------
/dev/nvme0n1     SDM00000ECB8         UCSC-F-H16003                            1           1.60
  TB /   1.60  TB    512   B +  0 B   KNCCP100
/dev/nvme1n1     8948c0b5ae4d4579     Linux                                    2         268.44
  GB / 268.44  GB    512   B +  0 B   5.2.2-1.
[root@Initiator_2 ~]# █
```

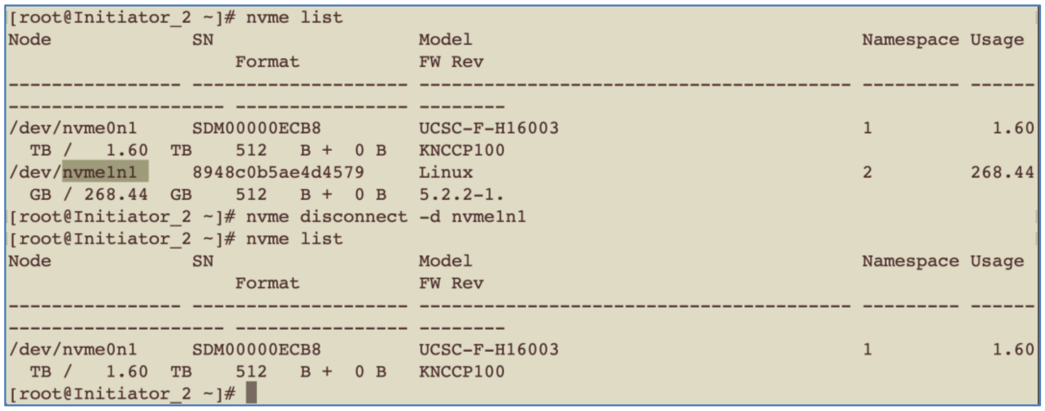
- NVME Connectnvme discover **-t rdma -a 10.254.108.2 -n <subnqn.number>**.

  WhenTarget is connected, it shows up under '**nvmelist**' output:



```
[root@Initiator_2 ~]# nvme connect -t rdma -a 10.254.108.2 -n nqn.2014-08.org.nvmexpress:NVMf:u
uid:a7dc6e92-ec2f-4d08-9d63-e45346c6ad80
[root@Initiator_2 ~]# nvme list
Node             SN                   Model                                    Namespace Usage
                     Format           FW Rev
---------------- -------------------- ---------------------------------------- --------- ------
-------------------- ---------------- --------
/dev/nvme0n1     SDM00000ECB8         UCSC-F-H16003                            1           1.60
  TB /   1.60  TB    512   B +  0 B   KNCCP100
/dev/nvme1n1     8948c0b5ae4d4579     Linux                                    2         268.44
  GB / 268.44  GB    512   B +  0 B   5.2.2-1.
[root@Initiator_2 ~]# █
```

- NVME Disconnect **nvme discover <nvme_drive_number>**



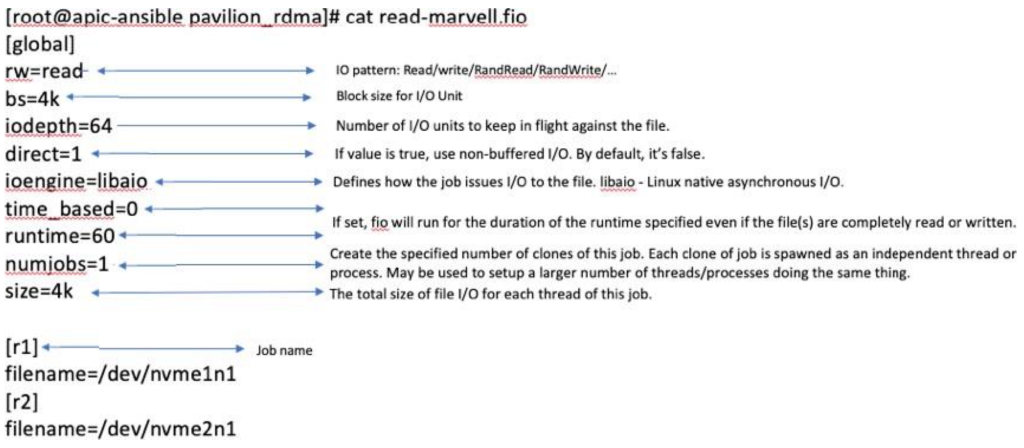
```
[root@Initiator_2 ~]# nvme list
Node             SN                   Model                                    Namespace Usage
                      Format           FW Rev
---------------- -------------------- ---------------------------------------- --------- ------
-------------------- ---------------- --------
/dev/nvme0n1     SDM00000ECB8         UCSC-F-H16003                            1           1.60
  TB /   1.60  TB    512   B +  0 B   KNCCP100
/dev/nvme1n1     8948c0b5ae4d4579     Linux                                    2         268.44
  GB / 268.44  GB    512   B +  0 B   5.2.2-1.
[root@Initiator_2 ~]# nvme disconnect -d nvme1n1
[root@Initiator_2 ~]# nvme list
Node             SN                   Model                                    Namespace Usage
                      Format           FW Rev
---------------- -------------------- ---------------------------------------- --------- ------
-------------------- ---------------- --------
/dev/nvme0n1     SDM00000ECB8         UCSC-F-H16003                            1           1.60
  TB /   1.60  TB    512   B +  0 B   KNCCP100
[root@Initiator_2 ~]#
```

FIO - Flexible I/O Storage Test tool:

FIO is an open-source, flexible I/O measuring tool that supports multiple threads, various queue depths, different measuring test duration, and many other parameters. FIO is an industry-standard tool for measuring storage performance in Linux. It can be run either by command line or configure file. In our lab, we used Configure file to run FIO tests.

FIO parameters:

The FIO configuration below is an example describing how to define and set a parameter. Certainly, the parameters in the file are the most popular ones but not all. For the entire parameter definition, see https://fio.readthedocs.io/en/latest/.



```
[root@apic-ansible pavilion_rdma]# cat read-marvell.fio
[global]
rw=read
bs=4k
iodepth=64
direct=1
ioengine=libaio
time_based=0
runtime=60
numjobs=1
size=4k

[r1]
filename=/dev/nvme1n1
[r2]
filename=/dev/nvme2n1
```

rw=read	IO pattern: Read/write/RandRead/RandWrite/…
bs=4k	Block size for I/O Unit
iodepth=64	Number of I/O units to keep in flight against the file.
direct=1	If value is true, use non-buffered I/O. By default, it's false.
ioengine=libaio	Defines how the job issues I/O to the file. libaio - Linux native asynchronous I/O.
time_based=0 / runtime=60	If set, fio will run for the duration of the runtime specified even if the file(s) are completely read or written.
numjobs=1	Create the specified number of clones of this job. Each clone of job is spawned as an independent thread or process. May be used to setup a larger number of threads/processes doing the same thing.
size=4k	The total size of file I/O for each thread of this job.
[r1]	Job name

FIO output example – sequential read

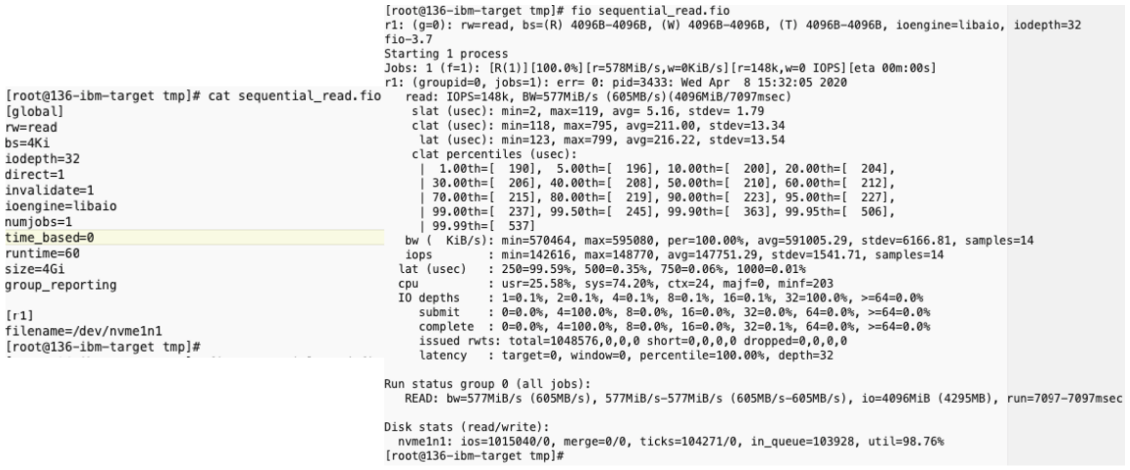
On the left of the screenshot below, it defines that the FIO test is file size based ("size=4Gi"), instead of the duration based ("time_based=0"). The read/write pattern is sequential read ("rw=read");

On the right of the screenshot below, it shows that the IOPS for job r1 is 148K per second; the average slat (submission latency) is 5.16 usec, the avg clat (completion latency) is 211 usec and the total avg latency is 216.22 usec; moreover, the CPU user and system usage are shown in the output; at the bottom of the output, the bw, io size, and running time are also shown. As for the clat percentile part, it means that the usec latency is associated with the x-th percentile.

slat – submission latency: this is the time that it took to submit I/O.

clat – completion latency: it is the time from submission to completion I/O.

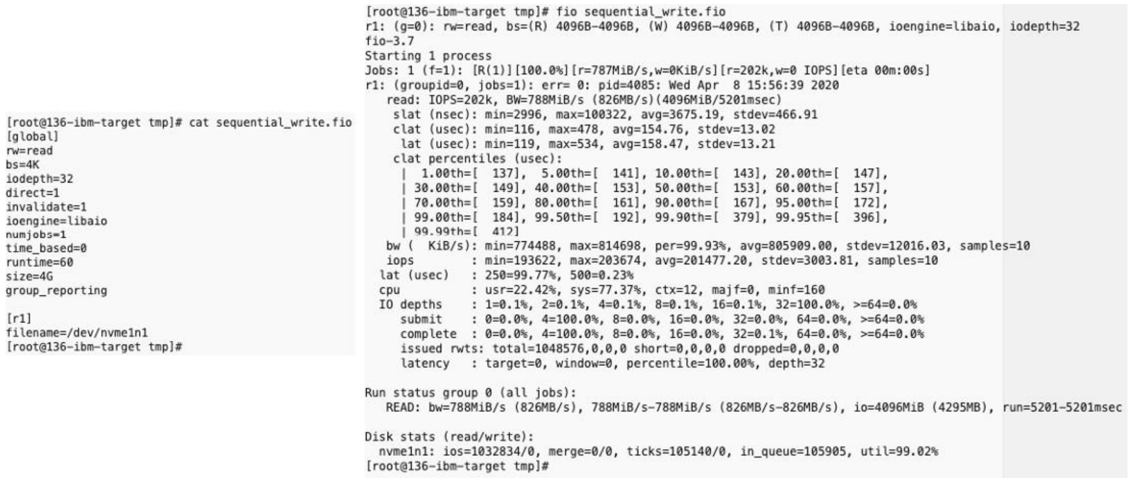
lat – total latency: it's the time from when the FIO created I/O to completion of the I/O operation.



```
[root@136-ibm-target tmp]# cat sequential_read.fio
[global]
rw=read
bs=4Ki
iodepth=32
direct=1
invalidate=1
ioengine=libaio
numjobs=1
time_based=0
runtime=60
size=4Gi
group_reporting

[r1]
filename=/dev/nvme1n1
[root@136-ibm-target tmp]#
```

```
[root@136-ibm-target tmp]# fio sequential_read.fio
r1: (g=0): rw=read, bs=(R) 4096B-4096B, (W) 4096B-4096B, (T) 4096B-4096B, ioengine=libaio, iodepth=32
fio-3.7
Starting 1 process
Jobs: 1 (f=1): [R(1)][100.0%][r=578MiB/s,w=0KiB/s][r=148k,w=0 IOPS][eta 00m:00s]
r1: (groupid=0, jobs=1): err= 0: pid=3433: Wed Apr  8 15:32:05 2020
   read: IOPS=148k, BW=577MiB/s (605MB/s)(4096MiB/7097msec)
    slat (usec): min=2, max=119, avg= 5.16, stdev= 1.79
    clat (usec): min=118, max=795, avg=211.00, stdev=13.34
     lat (usec): min=123, max=799, avg=216.22, stdev=13.54
    clat percentiles (usec):
     |  1.00th=[  190],  5.00th=[  196], 10.00th=[  200], 20.00th=[  204],
     | 30.00th=[  206], 40.00th=[  208], 50.00th=[  210], 60.00th=[  212],
     | 70.00th=[  215], 80.00th=[  219], 90.00th=[  223], 95.00th=[  227],
     | 99.00th=[  237], 99.50th=[  245], 99.90th=[  363], 99.95th=[  506],
     | 99.99th=[  537]
   bw (  KiB/s): min=570464, max=595080, per=100.00%, avg=591005.29, stdev=6166.81, samples=14
   iops        : min=142616, max=148770, avg=147751.29, stdev=1541.71, samples=14
  lat (usec)   : 250=99.59%, 500=0.35%, 750=0.06%, 1000=0.01%
  cpu          : usr=25.58%, sys=74.20%, ctx=24, majf=0, minf=203
  IO depths    : 1=0.1%, 2=0.1%, 4=0.1%, 8=0.1%, 16=0.1%, 32=100.0%, >=64=0.0%
     submit    : 0=0.0%, 4=100.0%, 8=0.0%, 16=0.0%, 32=0.0%, 64=0.0%, >=64=0.0%
     complete  : 0=0.0%, 4=100.0%, 8=0.0%, 16=0.0%, 32=0.1%, 64=0.0%, >=64=0.0%
     issued rwts: total=1048576,0,0,0 short=0,0,0,0 dropped=0,0,0,0
     latency   : target=0, window=0, percentile=100.00%, depth=32

Run status group 0 (all jobs):
   READ: bw=577MiB/s (605MB/s), 577MiB/s-577MiB/s (605MB/s-605MB/s), io=4096MiB (4295MB), run=7097-7097msec

Disk stats (read/write):
  nvme1n1: ios=1015040/0, merge=0/0, ticks=104271/0, in_queue=103928, util=98.76%
[root@136-ibm-target tmp]#
```

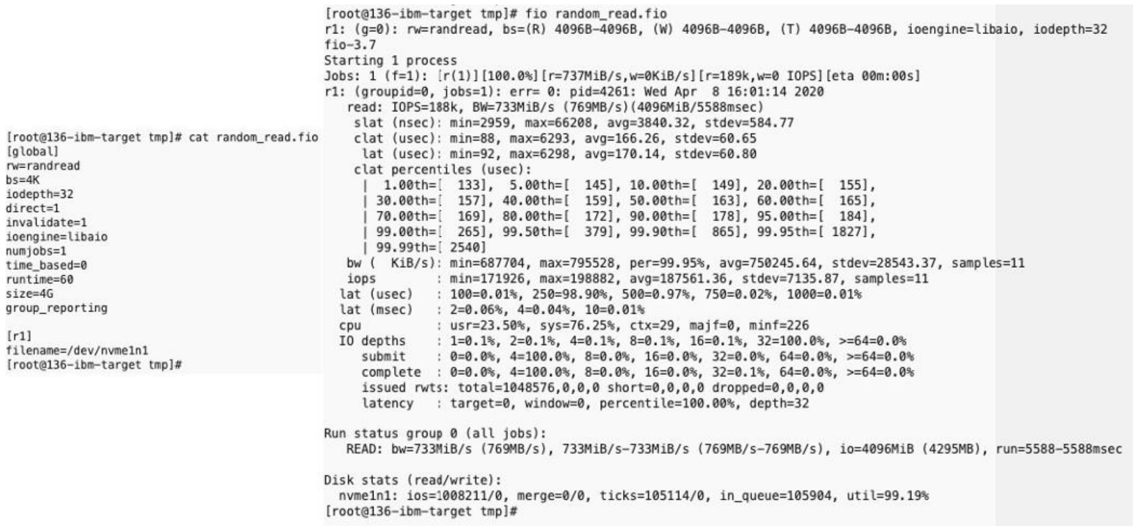
FIO output example – sequential write



```
[root@136-ibm-target tmp]# cat sequential_write.fio
[global]
rw=read
bs=4K
iodepth=32
direct=1
invalidate=1
ioengine=libaio
numjobs=1
time_based=0
runtime=60
size=4G
group_reporting

[r1]
filename=/dev/nvme1n1
[root@136-ibm-target tmp]#
```

```
[root@136-ibm-target tmp]# fio sequential_write.fio
r1: (g=0): rw=read, bs=(R) 4096B-4096B, (W) 4096B-4096B, (T) 4096B-4096B, ioengine=libaio, iodepth=32
fio-3.7
Starting 1 process
Jobs: 1 (f=1): [R(1)][100.0%][r=787MiB/s,w=0KiB/s][r=202k,w=0 IOPS][eta 00m:00s]
r1: (groupid=0, jobs=1): err= 0: pid=4085: Wed Apr  8 15:56:39 2020
   read: IOPS=202k, BW=788MiB/s (826MB/s)(4096MiB/5201msec)
    slat (nsec): min=2996, max=100322, avg=3675.19, stdev=466.91
    clat (usec): min=116, max=478, avg=154.76, stdev=13.02
     lat (usec): min=119, max=534, avg=158.47, stdev=13.21
    clat percentiles (usec):
     |  1.00th=[  137],  5.00th=[  141], 10.00th=[  143], 20.00th=[  147],
     | 30.00th=[  149], 40.00th=[  153], 50.00th=[  153], 60.00th=[  157],
     | 70.00th=[  159], 80.00th=[  161], 90.00th=[  167], 95.00th=[  172],
     | 99.00th=[  184], 99.50th=[  192], 99.90th=[  379], 99.95th=[  396],
     | 99.99th=[  412]
   bw (  KiB/s): min=774488, max=814698, per=99.93%, avg=805909.00, stdev=12016.03, samples=10
   iops        : min=193622, max=203674, avg=201477.20, stdev=3003.81, samples=10
  lat (usec)   : 250=99.77%, 500=0.23%
  cpu          : usr=22.42%, sys=77.37%, ctx=12, majf=0, minf=160
  IO depths    : 1=0.1%, 2=0.1%, 4=0.1%, 8=0.1%, 16=0.1%, 32=100.0%, >=64=0.0%
     submit    : 0=0.0%, 4=100.0%, 8=0.0%, 16=0.0%, 32=0.0%, 64=0.0%, >=64=0.0%
     complete  : 0=0.0%, 4=100.0%, 8=0.0%, 16=0.0%, 32=0.1%, 64=0.0%, >=64=0.0%
     issued rwts: total=1048576,0,0,0 short=0,0,0,0 dropped=0,0,0,0
     latency   : target=0, window=0, percentile=100.00%, depth=32

Run status group 0 (all jobs):
   READ: bw=788MiB/s (826MB/s), 788MiB/s-788MiB/s (826MB/s-826MB/s), io=4096MiB (4295MB), run=5201-5201msec

Disk stats (read/write):
  nvme1n1: ios=1032834/0, merge=0/0, ticks=105140/0, in_queue=105905, util=99.02%
[root@136-ibm-target tmp]#
```

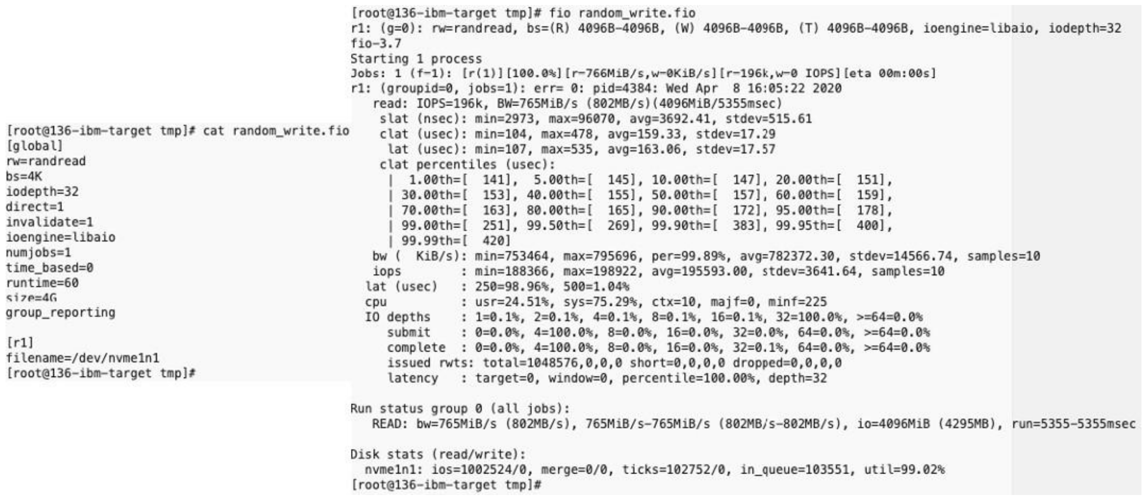
FIO output example – random read



```
[root@136-ibm-target tmp]# cat random_read.fio
[global]
rw=randread
bs=4K
iodepth=32
direct=1
invalidate=1
ioengine=libaio
numjobs=1
time_based=0
runtime=60
size=4G
group_reporting

[r1]
filename=/dev/nvme1n1
[root@136-ibm-target tmp]#
```

```
[root@136-ibm-target tmp]# fio random_read.fio
r1: (g=0): rw=randread, bs=(R) 4096B-4096B, (W) 4096B-4096B, (T) 4096B-4096B, ioengine=libaio, iodepth=32
fio-3.7
Starting 1 process
Jobs: 1 (f=1): [r(1)][100.0%][r=737MiB/s,w=0KiB/s][r=189k,w=0 IOPS][eta 00m:00s]
r1: (groupid=0, jobs=1): err= 0: pid=4261: Wed Apr  8 16:01:14 2020
   read: IOPS=188k, BW=733MiB/s (769MB/s)(4096MiB/5588msec)
    slat (nsec): min=2959, max=66208, avg=3840.32, stdev=584.77
    clat (usec): min=88, max=6293, avg=166.26, stdev=60.65
     lat (usec): min=92, max=6298, avg=170.14, stdev=60.80
    clat percentiles (usec):
     |  1.00th=[  133],  5.00th=[  145], 10.00th=[  149], 20.00th=[  155],
     | 30.00th=[  157], 40.00th=[  159], 50.00th=[  163], 60.00th=[  165],
     | 70.00th=[  169], 80.00th=[  172], 90.00th=[  178], 95.00th=[  184],
     | 99.00th=[  265], 99.50th=[  379], 99.90th=[  865], 99.95th=[ 1827],
     | 99.99th=[ 2540]
   bw (  KiB/s): min=687704, max=795528, per=99.95%, avg=750245.64, stdev=28543.37, samples=11
   iops        : min=171926, max=198882, avg=187561.36, stdev=7135.87, samples=11
  lat (usec)   : 100=0.01%, 250=98.90%, 500=0.97%, 750=0.02%, 1000=0.01%
  lat (msec)   : 2=0.06%, 4=0.04%, 10=0.01%
  cpu          : usr=23.50%, sys=76.25%, ctx=29, majf=0, minf=226
  IO depths    : 1=0.1%, 2=0.1%, 4=0.1%, 8=0.1%, 16=0.1%, 32=100.0%, >=64=0.0%
     submit    : 0=0.0%, 4=100.0%, 8=0.0%, 16=0.0%, 32=0.0%, 64=0.0%, >=64=0.0%
     complete  : 0=0.0%, 4=100.0%, 8=0.0%, 16=0.0%, 32=0.1%, 64=0.0%, >=64=0.0%
     issued rwts: total=1048576,0,0,0 short=0,0,0,0 dropped=0,0,0,0
     latency   : target=0, window=0, percentile=100.00%, depth=32

Run status group 0 (all jobs):
   READ: bw=733MiB/s (769MB/s), 733MiB/s-733MiB/s (769MB/s-769MB/s), io=4096MiB (4295MB), run=5588-5588msec

Disk stats (read/write):
  nvme1n1: ios=1008211/0, merge=0/0, ticks=105114/0, in_queue=105904, util=99.19%
[root@136-ibm-target tmp]#
```

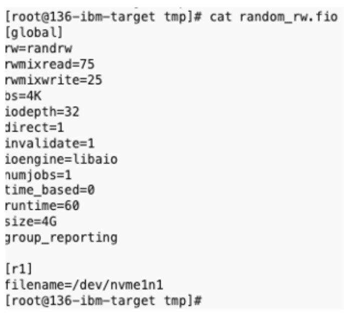
FIO output example – random write



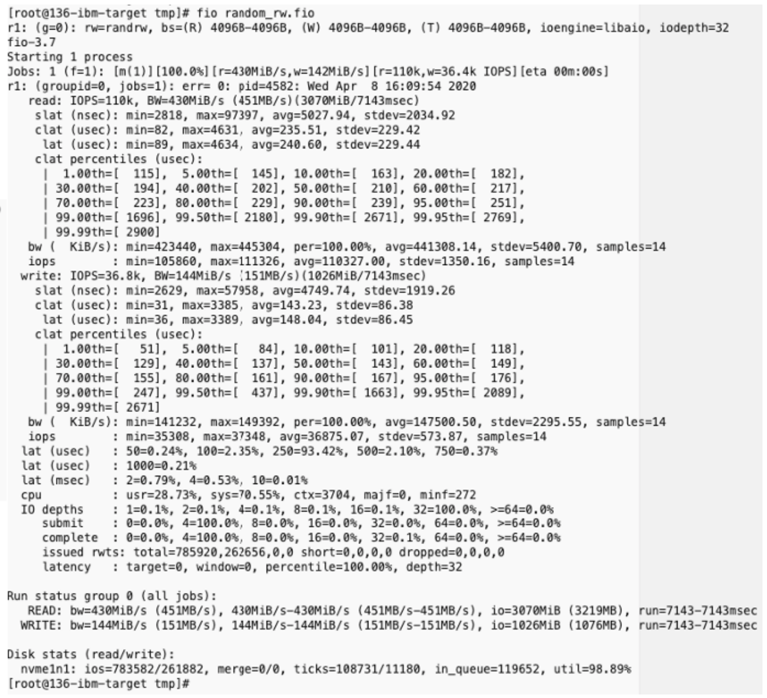
```
[root@136-ibm-target tmp]# cat random_write.fio
[global]
rw=randread
bs=4K
iodepth=32
direct=1
invalidate=1
ioengine=libaio
numjobs=1
time_based=0
runtime=60
size=4G
group_reporting

[r1]
filename=/dev/nvme1n1
[root@136-ibm-target tmp]#
```

```
[root@136-ibm-target tmp]# fio random_write.fio
r1: (g=0): rw=randread, bs=(R) 4096B-4096B, (W) 4096B-4096B, (T) 4096B-4096B, ioengine=libaio, iodepth=32
fio-3.7
Starting 1 process
Jobs: 1 (f-1): [r(1)][100.0%][r-766MiB/s,w-0KiB/s][r-196k,w-0 IOPS][eta 00m:00s]
r1: (groupid=0, jobs=1): err= 0: pid=4384: Wed Apr  8 16:05:22 2020
   read: IOPS=196k, BW=765MiB/s (802MB/s)(4096MiB/5355msec)
    slat (nsec): min=2973, max=96070, avg=3692.41, stdev=515.61
    clat (usec): min=104, max=478, avg=159.33, stdev=17.29
     lat (usec): min=107, max=535, avg=163.06, stdev=17.57
    clat percentiles (usec):
     |  1.00th=[  141],  5.00th=[  145], 10.00th=[  147], 20.00th=[  151],
     | 30.00th=[  153], 40.00th=[  155], 50.00th=[  157], 60.00th=[  159],
     | 70.00th=[  163], 80.00th=[  165], 90.00th=[  172], 95.00th=[  178],
     | 99.00th=[  251], 99.50th=[  269], 99.90th=[  383], 99.95th=[  400],
     | 99.99th=[  420]
   bw (  KiB/s): min=753464, max=795696, per=99.89%, avg=782372.30, stdev=14566.74, samples=10
   iops        : min=188366, max=198922, avg=195593.00, stdev=3641.64, samples=10
  lat (usec)   : 250=98.96%, 500=1.04%
  cpu          : usr=24.51%, sys=75.29%, ctx=10, majf=0, minf=225
  IO depths    : 1=0.1%, 2=0.1%, 4=0.1%, 8=0.1%, 16=0.1%, 32=100.0%, >=64=0.0%
     submit    : 0=0.0%, 4=100.0%, 8=0.0%, 16=0.0%, 32=0.0%, 64=0.0%, >=64=0.0%
     complete  : 0=0.0%, 4=100.0%, 8=0.0%, 16=0.0%, 32=0.1%, 64=0.0%, >=64=0.0%
     issued rwts: total=1048576,0,0,0 short=0,0,0,0 dropped=0,0,0,0
     latency   : target=0, window=0, percentile=100.00%, depth=32

Run status group 0 (all jobs):
   READ: bw=765MiB/s (802MB/s), 765MiB/s-765MiB/s (802MB/s-802MB/s), io=4096MiB (4295MB), run=5355-5355msec

Disk stats (read/write):
  nvme1n1: ios=1002524/0, merge=0/0, ticks=102752/0, in_queue=103551, util=99.02%
[root@136-ibm-target tmp]#
```

FIO output example – random read/write



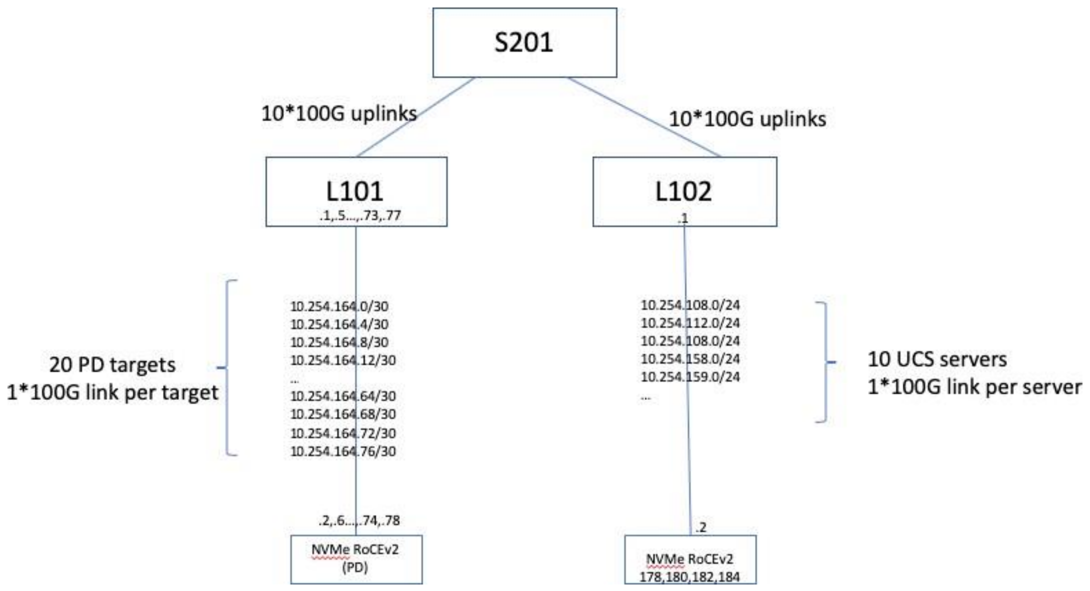
```
[root@136-ibm-target tmp]# cat random_rw.fio
[global]
rw=randrw
rwmixread=75
rwmixwrite=25
bs=4K
iodepth=32
direct=1
invalidate=1
ioengine=libaio
numjobs=1
time_based=0
runtime=60
size=4G
group_reporting

[r1]
filename=/dev/nvme1n1
[root@136-ibm-target tmp]#
```



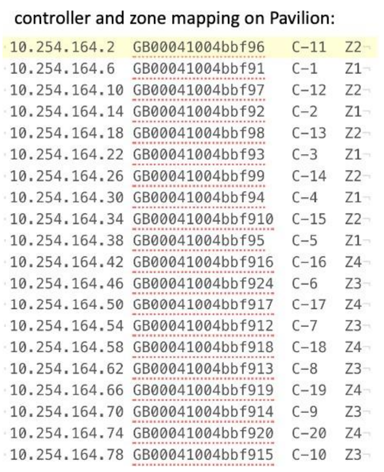
```
[root@136-ibm-target tmp]# fio random_rw.fio
r1: (g=0): rw=randrw, bs=(R) 4096B-4096B, (W) 4096B-4096B, (T) 4096B-4096B, ioengine=libaio, iodepth=32
fio-3.7
Starting 1 process
Jobs: 1 (f=1): [m(1)][100.0%][r=430MiB/s,w=142MiB/s][r=110k,w=36.4k IOPS][eta 00m:00s]
r1: (groupid=0, jobs=1): err= 0: pid=4582: Wed Apr  8 16:09:54 2020
   read: IOPS=110k, BW=430MiB/s (451MB/s)(3070MiB/7143msec)
    slat (nsec): min=2818, max=97397, avg=5027.94, stdev=2034.92
    clat (usec): min=82, max=4631, avg=235.51, stdev=229.42
     lat (usec): min=89, max=4634, avg=240.60, stdev=229.44
    clat percentiles (usec):
     |  1.00th=[  115],  5.00th=[  145], 10.00th=[  163], 20.00th=[  182],
     | 30.00th=[  194], 40.00th=[  202], 50.00th=[  210], 60.00th=[  217],
     | 70.00th=[  223], 80.00th=[  229], 90.00th=[  239], 95.00th=[  251],
     | 99.00th=[ 1696], 99.50th=[ 2180], 99.90th=[ 2671], 99.95th=[ 2769],
     | 99.99th=[ 2900]
   bw (  KiB/s): min=423440, max=445304, per=100.00%, avg=441308.14, stdev=5400.70, samples=14
   iops        : min=105860, max=111326, avg=110327.00, stdev=1350.16, samples=14
  write: IOPS=36.8k, BW=144MiB/s (151MB/s)(1026MiB/7143msec)
    slat (nsec): min=2629, max=57958, avg=4749.74, stdev=1919.26
    clat (usec): min=31, max=3385, avg=143.23, stdev=86.38
     lat (usec): min=36, max=3389, avg=148.04, stdev=86.45
    clat percentiles (usec):
     |  1.00th=[   51],  5.00th=[   84], 10.00th=[  101], 20.00th=[  118],
     | 30.00th=[  129], 40.00th=[  137], 50.00th=[  143], 60.00th=[  149],
     | 70.00th=[  155], 80.00th=[  161], 90.00th=[  167], 95.00th=[  176],
     | 99.00th=[  247], 99.50th=[  437], 99.90th=[ 1663], 99.95th=[ 2089],
     | 99.99th=[ 2671]
   bw (  KiB/s): min=141232, max=149392, per=100.00%, avg=147500.50, stdev=2295.55, samples=14
   iops        : min=35308, max=37348, avg=36875.07, stdev=573.87, samples=14
  lat (usec)   : 50=0.24%, 100=2.35%, 250=93.42%, 500=2.10%, 750=0.37%
  lat (usec)   : 1000=0.21%
  lat (msec)   : 2=0.79%, 4=0.53%, 10=0.01%
  cpu          : usr=28.73%, sys=70.55%, ctx=3704, majf=0, minf=272
  IO depths    : 1=0.1%, 2=0.1%, 4=0.1%, 8=0.1%, 16=0.1%, 32=100.0%, >=64=0.0%
     submit    : 0=0.0%, 4=100.0%, 8=0.0%, 16=0.0%, 32=0.0%, 64=0.0%, >=64=0.0%
     complete  : 0=0.0%, 4=100.0%, 8=0.0%, 16=0.0%, 32=0.1%, 64=0.0%, >=64=0.0%
     issued rwts: total=785920,262656,0,0 short=0,0,0,0 dropped=0,0,0,0
     latency   : target=0, window=0, percentile=100.00%, depth=32

Run status group 0 (all jobs):
   READ: bw=430MiB/s (451MB/s), 430MiB/s-430MiB/s (451MB/s-451MB/s), io=3070MiB (3219MB), run=7143-7143msec
  WRITE: bw=144MiB/s (151MB/s), 144MiB/s-144MiB/s (151MB/s-151MB/s), io=1026MiB (1076MB), run=7143-7143msec

Disk stats (read/write):
  nvme1n1: ios=783582/261882, merge=0/0, ticks=108731/11180, in_queue=119652, util=98.89%
[root@136-ibm-target tmp]#
```

Running NVMe RoCE performance test with FIO:

For measuring the NVMe RoCE performance over Cisco N9K switches, we built up the spine-leaf topology as below: Spine switch is N9K-C9332C and Leaf switch is N9K-C9336C-FX2; the initiator is Cisco UCS server (C240-M5) and the target is Pavilion Data's Rack Scale Storage platform (fully-loaded with 72 Samsung SSDs); all links are connected by 100G-based fiber cables. For more detailed specifications, see Cisco and Pavilion documentation, respectively.

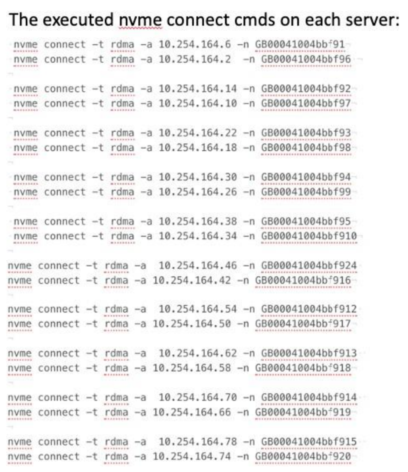


For maximizing the performance, each UCS initiator connects to 2 pavilion targets and meanwhile each target is located in a different zone so that there is no oversubscription on the 100G link between the server and the switch. For detailed connection mapping, see the following images.



controller and zone mapping on Pavilion:

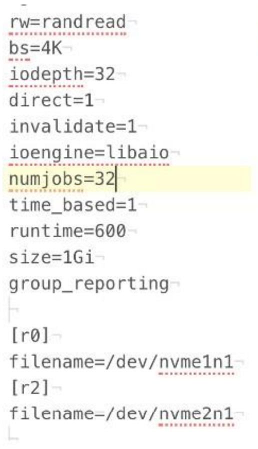
```
10.254.164.2  GB00041004bbf96   C-11  Z2
10.254.164.6  GB00041004bbf91   C-1   Z1
10.254.164.10 GB00041004bbf97   C-12  Z2
10.254.164.14 GB00041004bbf92   C-2   Z1
10.254.164.18 GB00041004bbf98   C-13  Z2
10.254.164.22 GB00041004bbf93   C-3   Z1
10.254.164.26 GB00041004bbf99   C-14  Z2
10.254.164.30 GB00041004bbf94   C-4   Z1
10.254.164.34 GB00041004bbf910  C-15  Z2
10.254.164.38 GB00041004bbf95   C-5   Z1
10.254.164.42 GB00041004bbf916  C-16  Z4
10.254.164.46 GB00041004bbf924  C-6   Z3
10.254.164.50 GB00041004bbf917  C-17  Z4
10.254.164.54 GB00041004bbf912  C-7   Z3
10.254.164.58 GB00041004bbf918  C-18  Z4
10.254.164.62 GB00041004bbf913  C-8   Z3
10.254.164.66 GB00041004bbf919  C-19  Z4
10.254.164.70 GB00041004bbf914  C-9   Z3
10.254.164.74 GB00041004bbf920  C-20  Z4
10.254.164.78 GB00041004bbf915  C-10  Z3
```



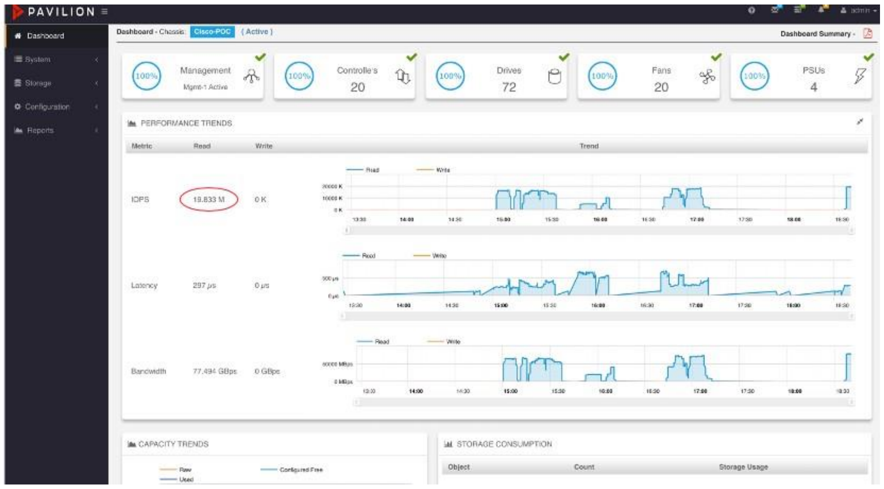
The executed nvme connect cmds on each server:

```
 nvme connect -t rdma -a 10.254.164.6 -n GB00041004bbf91
 nvme connect -t rdma -a 10.254.164.2  -n GB00041004bbf96

 nvme connect -t rdma -a 10.254.164.14 -n GB00041004bbf92
 nvme connect -t rdma -a 10.254.164.10 -n GB00041004bbf97

 nvme connect -t rdma -a 10.254.164.22 -n GB00041004bbf93
 nvme connect -t rdma -a 10.254.164.18 -n GB00041004bbf98

 nvme connect -t rdma -a 10.254.164.30 -n GB00041004bbf94
 nvme connect -t rdma -a 10.254.164.26 -n GB00041004bbf99

 nvme connect -t rdma -a 10.254.164.38 -n GB00041004bbf95
 nvme connect -t rdma -a 10.254.164.34 -n GB00041004bbf910

nvme connect -t rdma -a  10.254.164.46 -n GB00041004bbf924
nvme connect -t rdma -a 10.254.164.42 -n GB00041004bbf916

nvme connect -t rdma -a  10.254.164.54 -n GB00041004bbf912
nvme connect -t rdma -a 10.254.164.50 -n GB00041004bbf917

nvme connect -t rdma -a  10.254.164.62 -n GB00041004bbf913
nvme connect -t rdma -a 10.254.164.58 -n GB00041004bbf918

nvme connect -t rdma -a  10.254.164.70 -n GB00041004bbf914
nvme connect -t rdma -a 10.254.164.66 -n GB00041004bbf919

nvme connect -t rdma -a  10.254.164.78 -n GB00041004bbf915
nvme connect -t rdma -a 10.254.164.74 -n GB00041004bbf920
```

The NMVe RoCE test result is shown and monitored on the Pavilion GUI dashboard shown below. Meanwhile, the FIO configuration file is shown for reference.



```
rw=randread
bs=4K
iodepth=32
direct=1
invalidate=1
ioengine=libaio
numjobs=32
time_based=1
runtime=600
size=1Gi
group_reporting

[r0]
filename=/dev/nvme1n1
[r2]
filename=/dev/nvme2n1
```



Metric	Read	Write	Trend
IOPS	19.833 M	0 K	
Latency	297 µs	0 µs	
Bandwidth	77.494 GBps	0 GBps	

Object	Count	Storage Usage

Reference

VXLAN Multi-Site Cisco White Paper

https://www.cisco.com/c/en/us/products/collateral/switches/nexus-9000-series-switches/white-paper-c11-739942.html

Intelligent Buffer Management on Cisco Nexus 9000 Series Switches White Paper

https://www.cisco.com/c/en/us/products/collateral/switches/nexus-9000-series-switches/white-paper-c11-738488.html

InfiniBand Architecture Specification

https://www.ciscolive.com/c/dam/r/ciscolive/us/docs/2019/pdf/BRKDCN-2213.pdf

NVME-oF Specification

https://nvmexpress.org/developers/nvme-of-specification/

nvme-CLI Command Reference

https://github.com/linux-nvme/nvme-cli

Flexible I/O Tester Reference

https://fio.readthedocs.io/en/latest/fio_doc.html

Pavilion GUI Configuration Guide

https://pavilion.io/support/

RDMA over Converged Ethernet (RoCE) on Cisco Nexus 9300

https://aboutnetworks.net/rocev2-on-nexus9k/

NVMe over Fabrics and RDMA for network engineers

https://aboutnetworks.net/nvme-and-nvmeof/

https://network.nvidia.com/pdf/solutions/benefits-of-RDMA-over-routed-fabrics.pdf

Legal Information