SANS Institute
Information Security Reading Room

Network Segmentation of Users on Multi-User Servers and Networks

Ryan Cox

Copyright SANS Institute 2021. Author Retains Full Rights.

This paper is from the SANS Institute Reading Room site. Reposting is not permitted without express written permission.

https://t.me/learningnets

Network Segmentation of Users on Multi-User Servers and Networks

GIAC (GCUX) Gold Certification

Author: Ryan Cox, ryan_cox@byu.edu
Advisor: Russell Eubanks

Accepted: December 22, 2020

Abstract

In High Performance Computing (HPC) environments, hundreds of users can be logged in and running batch jobs simultaneously on clusters of servers in a multi-user environment. Security controls may be in place for much of the overall HPC environment, but user network communication is rarely included in those controls. Some users run software that must listen on arbitrary network ports, exposing user software to attacks by others. This creates the possibility of account compromise by fellow users who have access to those same servers and networks. A solution was developed to transparently segregate users from each other both locally and over the network. The result is easy to install and administer.

© 2021 The SANS Institute, Author Retains Full Rights
https://t.me/learningnets
© 2021 The SANS Institute Author retains full rights.

1. Introduction

1.1. Background on High Performance Computing

High Performance Computing (HPC) environments are relatively porous from a security perspective. By design, users can typically develop, download, compile, and execute arbitrary code, often with no code review or oversight whatsoever. Some of that code communicates over networks and may listen on various network ports.

Hundreds of users may be actively logged in or running work at the same time across hundreds or thousands of servers, even at a moderately-sized HPC site. Almost all HPC centers use a batch job scheduling system such as Slurm[1] to manage access to clusters of servers (a.k.a. "nodes" in HPC parlance) that run Linux (Trader, 2019). Users can typically log in directly over ssh to nodes on which their jobs are assigned.

Many HPC sites store valuable information that may be of interest to attackers. For an HPC site in the United States, valuable targets may include Confidential Unclassified Information (CUI)—much of which is export-controlled—from the Department of Defense, Department of Energy, and other government entities[2]; Protected Health Information covered by HIPAA; valuable intellectual property; and other forms of protected data.

This project aims to improve the security of network communications by users of HPC systems by creating per-user virtual networks. As with any security solution, defense-in-depth is required for the best chance of defeating attackers.

1.2. Network Communication in HPC

HPC networks can be secured against malicious external network traffic but measures are rarely taken to prevent users from connecting to network ports opened by other users' applications. If vulnerable due to misconfiguration or software flaws, these

[1] Slurm manages batch job submission, prioritization, scheduling, job launch on nodes, accounting, etc. (SchedMD LLC, n.d.-a.).

[2] NIST Special Publication 800-171, DFARS 252.204-7012, the Department of Defense CMMC program, and other standards govern the protection of some of this data. Export-control restrictions may include ITAR, EAR, OFAC, or other more restrictive requirements.

© 2021 The SANS Institute https://t.me/learningnets Author retains full rights.

listening applications can be compromised by other users of the HPC system across the network or on localhost. Unfortunately, this is a difficult problem to solve because of the nature of HPC environments.

In HPC environments, many users need to execute code across multiple nodes as part of a single batch job, typically using a library such as Message Passing Interface (MPI). Most MPI implementations open several arbitrary network ports that must be available between nodes, thus complicating even basic firewalling attempts[3] (The Open MPI Project, 2019).

Some software is intentionally made available over the network by users, such as database server instances launched by the users themselves. Users may even run web servers for various purposes, such as Jupyter Notebook[4] (Project Jupyter, n.d.). The software is typically intended to be available only to the user who launched it, but security controls are often non-existent. Additionally, software is rarely updated once the user installs it, thus ensuring that it will not receive security patches[5].

An external attacker could compromise a user's account, through phishing for credentials or some other method, and then use that account to find vulnerable software that was launched by other users and is listening across the network or on localhost on various nodes. From there, the attacker could exploit the software to gain access to a different victim user's account. A sufficiently motivated attacker—such as an Advanced Persistent Threat (APT) actor—may attack multiple user accounts until valuable data[6] is located.

---

[3] Ports can usually be selected ahead of time for ease of writing firewall rules, but the complexities of managing which users can open which ports may make this an overly difficult solution for most.

[4] Jupyter Notebook is a popular web-based IDE for data science, scientific computing, and machine learning (Project Jupyter, n.d.).

[5] Some users are required to conduct research in a completely reproducible manner, including keeping the specific software stack static (Sandve et al., 2013). Even if that is not a constraint for a researcher, researchers rarely worry about security since their focus is on the research itself.

[6] Valuable data may include protected data mentioned in the previous section or other data of interest. Additionally, the victim may maintain ssh keys that could be used by the attacker for "island hopping". For example, an attacker may use a victim's ssh private key to access remote systems that allow access based on that key.

© 2021 The SANS Institute https://t.me/learningnets Author retains full rights.

1.3. Solution

This paper examines an approach to protecting users' networked applications from attacks by other users both over the network and on localhost. There are several approaches to achieving this that are likely to be viable, but this work focuses on two particular technologies. Of the many solutions that were considered, the solution presented here is the least complicated one that achieves a high level of security and a low performance impact.

In the solution presented here, per-user overlay networks are created and used in such a way that users can only communicate within their own overlay network[7]. Virtual Routing and Forwarding (VRF) configurations isolate users from each other *within* a system while Virtual Extensible LAN (VXLAN) networks isolate users from each other *between* systems. The solution is transparent to users. Performance was a key consideration in the design; benchmarks demonstrate this approach's viability. Complexity was also a major consideration since a solution that is difficult to understand will likely not be adopted by HPC centers.

The entire solution is simple to set up and consists of a single bash script, an additional line in a PAM configuration file, and a simple plugin for Slurm[8].

2. Constraints

The solution presented in this paper was developed with many constraints and assumptions that seem reasonable to the author. If not for these constraints, the certain replies of "containers", "namespaces", "virtual machines", "SDN", or other technologies from DevOps and enterprise IT administrators would likely be warranted.

Some of these constraints are based on the author's knowledge of typical HPC environments, the desire to minimize the overall complexity of the solution, various limitations of existing technologies, and the impact of the solution on performance.

[7] This applies only to "local" administrator-defined subnets. Users can still communicate outside of these networks if allowed to do so by routing and firewall configurations. See Section 4.2.
[8] The author focused on Slurm, but a similar approach should be easily adaptable to other job schedulers.

© 2021 The SANS Institute https://t.me/learningnets Author retains full rights.

These constraints are outlined in this section to explain why the solution described in this paper works the way that it does.

The solution should have a minimal impact on CPU, network bandwidth, and network latency. Any solution must also be easy for HPC systems administrators to understand and implement. Solutions that involve reconfiguring networking equipment are not necessary and add additional requirements to the installation, thus they were omitted from investigation. For example, Software-Defined Networking (SDN) was not investigated because it is uncommon in HPC and would require major architectural changes.

Point-to-point tunneling protocols are not viable. HPC batch jobs can run simultaneously on hundreds or thousands of nodes (Balaji et al., 2010); to create a high performance network configuration, a point-to-point protocol would require the assembly of a mesh network where each node explicitly creates a tunnel to each other node. Developing a solution to construct such a network seems overly complex and full of obstacles, especially when easier solutions already exist.

Most HPC local networks are considered "trusted", meaning that root accounts on each node on the network are controlled by HPC center staff and no end-user can access the network as a privileged user[9]. Encryption of network traffic within HPC networks is therefore not considered a useful feature due to the additional processor overhead it introduces.

2.1. Changes must be transparent to the user

The most limiting constraints relate to making all changes transparent to the user. All changes must be transparent when the user communicates on localhost, across a locally-attached subnet (referred to as "on-subnet" going forward), and to a remote network (referred to as "off-subnet" going forward).

A user must be able to connect to the network ports they open for their own services but not to the network ports opened by other users. Additionally, the user must

[9] System compromise is always a possibility, but the attacker would likely have more interesting targets to attack rather than jump between VXLANs.

© 2021 The SANS Institute https://t.me/learningnets Author retains full rights.

be able to access system services (e.g. sshd, Slurm, DNS if there is a local caching server, etc.) that are available on localhost through 127.0.0.1, ::1, and other local IP addresses.

The requirement to not impact on-subnet traffic is similar, but it introduces complexity depending on which solution is chosen. Network namespaces[10] would cause several problems due to the fact that sshd and Slurm must continue to be available to users on different nodes across local subnets. An instance of sshd could be configured to launch inside of each namespace, but Slurm is not currently capable of listening in multiple namespaces simultaneously or being launched in different namespaces on the same host. pam_slurm_adopt, a PAM module[11] for Slurm that handles ssh-launched processes, would be unable to function in an environment where ssh and Slurm are not listening in the same namespace (Cox, 2015).

Outbound communication to off-subnet hosts must not be impacted. However, inbound connections to user-opened ports from remote sources are intentionally not supported in this project.

3. Technologies

Many technologies were considered for this project. VXLAN was chosen as the network encapsulation protocol because it performs well, clearly achieves the design goals, and is easy to set up. For this use case, VRF interfaces were clearly superior to network namespaces for isolating users from each other within a

[10] Network namespaces "partition the use of the network—devices, addresses, ports, routes, firewall rules, etc.—into separate boxes, essentially virtualizing the network within a single running kernel instance." (Edge, 2014).

[11] Pluggable Authentication Modules (PAM) "is a suite of shared libraries that enable the local system administrator to choose how applications authenticate users" (Morgan & Kukuk, 2010). PAM allows systems administrators to choose authentication options, set up environments for logins, and many other custom actions. For example, the widely-adopted pam_slurm_adopt was written by this paper's author to verify that ssh connections to compute nodes are only made by users who have jobs assigned to that node (Cox, 2015). If the user has a job on the node, pam_slurm_adopt "adopts" the ssh process into the appropriate job for resource accounting and enforcement purposes.

© 2021 The SANS Institute https://t.me/learningnets Author retains full rights.

3.1. Virtual Extensible LAN (VXLAN)

This paper focuses on VXLAN as the preferred encapsulation technology. VXLAN was chosen because of its widespread availability and its many attractive features.

VXLAN is very similar in concept to VLANs, though it has some big differences (Mahalingam et al., 2014). VXLAN is a relatively simple protocol that encapsulates Ethernet packets in UDP packets and can be set up as a point-to-point tunnel or in a multicast group[12]. It has a 24-bit VXLAN Network Identifier (VNI) as opposed to the 12-bit VLAN Identifier (VID), resulting in over 16 million usable network identifiers compared to VLAN's 4,094 identifiers.

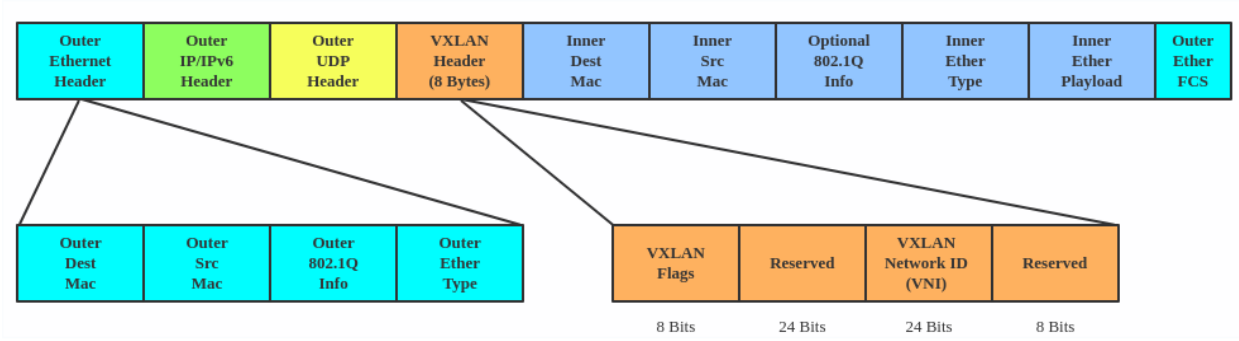

Figure 1. VXLAN packet headers (Liu, 2018).

VXLAN can use multicast to discover peers then transition to unicast after the discovery (Mahalingam et al., 2014). This results in minimal effort to create a working configuration while benefiting from unicast's performance. IP over InfiniBand (IPoIB)[13] and other non-Ethernet IP networks are supported since VXLAN is UDP-based and can function on any IP network.

For this project, each user is assigned a unique VNI based on the user's UID on Linux. This effectively creates a unique network for each user.

[12] GENEVE will purportedly supplant VXLAN and some other technologies some day (Schmaus, 2017), but it does not currently support multicast neighbor discovery on Linux like VXLAN does (Litvak, 2017).
[13] InfiniBand is a high speed, low latency RDMA network technology that is frequently used in HPC centers. IPoIB is an IP layer that runs on top of InfiniBand networks (Muehlfeld et al., n.d.).

© 2021 The SANS Institute https://t.me/learningnets Author retains full rights.

For a discussion of other technologies that were considered in place of VXLAN for network traffic segregation, see Appendix A.

3.2. Virtual Routing and Forwarding (VRF)

VRFs allow for separate routing tables to exist within the same system and for various processes to have different routes based on their VRF membership ("Virtual Routing and Forwarding (VRF)", n.d.). VRFs can be described as a method to present a process with a view of the network that is different from the views that other processes see. They only affect Layer 3 functionality; Layer 2 protocols such as LLDP are unaffected.

VRFs are useful in this solution due to their ability to present routes on different interfaces to different processes. A process inside a VRF can only use interfaces that are members of that VRF, and an interface can only belong to one VRF ("Virtual Routing and Forwarding (VRF)", n.d.).

VRFs and network namespaces can achieve some of the same goals, but they are distinctly different and have their own sets of advantages and disadvantages (Ahern, 2017). Many systems administrators are aware of network namespaces, though only a small number are likely to be aware of Linux VRFs. Network namespaces will be discussed throughout this section for comparison purposes.

VRFs provide network isolation between processes in different VRFs. A process in one VRF cannot talk to a listening port in a different VRF. The following output shows an example of using *ip vrf exec*[14] to place the netcat program (*nc*)[15] into two existing VRFs for various tests:

[14] *ip vrf exec* runs a command inside of a specified VRF
[15] *nc* can listen on or connect to network ports in order to receive or send arbitrary data

© 2021 The SANS Institute https://t.me/learningnets Author retains full rights.

```
[root@host ~]# ip vrf exec vrf9876 nc -l 5555 &
[1] 16841
[root@host ~]# echo HELLO | ip vrf exec vrf1234 nc 127.0.0.1 5555
Ncat: Connection refused.
[root@host ~]# echo HELLO | nc 127.0.0.1 5555
Ncat: Connection refused.
[root@host ~]# echo HELLO | ip vrf exec vrf9876 nc 127.0.0.1 5555
HELLO
[1]+  Done                    ip vrf exec vrf9876 nc -l 5555
[root@host ~]#
```

Figure 2. ip vrf exec invocation example

The example demonstrates that a process in vrf1234 cannot talk to a listening port opened by a process in vrf9876. Not even root-owned processes in the default VRF can connect to processes in those users' VRFs. If processes owned by UID 1234 are assigned to vrf1234 and processes from UID 9876 are assigned to vrf9876, users with UIDs 1234 and 9876 are isolated from each other and from any other users on the system. It should be noted that even stricter isolation can be easily achieved using network namespaces, but with significantly increased complexity in other areas.

A very useful feature of VRFs on Linux is the ability for unmodified applications to listen on all VRFs ("Virtual Routing and Forwarding (VRF)", n.d.). This optional feature is enabled by the sysctl knob net.ipv4.tcp_l3mdev_accept, along with similar knobs for udp and raw. Setting the value to "1" allows applications in the default VRF, such as sshd and Slurm's slurmd, to listen in all VRFs on the system automatically. This setting makes the services available to all local processes and inbound connections on any VRF from a remote system.

For example, this setting allows slurmd in the default VRF to handle connections on any interface from both localhost and remote hosts—subject to firewall configuration—regardless of whether remote connections enter through interface em1u1234 in vrf1234 or em1 in the default VRF. This is crucial to ensuring normal operation for "system" services that must be universally available.

The following example shows that a process in a VRF can only connect to sshd on localhost port 22 if net.ipv4.tcp_l3mdev_accept is set to "1":

© 2021 The SANS Institute https://t.me/learningnets Author retains full rights.

```
[root@host ~]# sysctl -w net.ipv4.tcp_l3mdev_accept=0
net.ipv4.tcp_l3mdev_accept = 0
[root@host ~]# ip vrf exec vrf1234 nc 127.0.0.1 22 </dev/null | head -1
Ncat: Connection refused.
[root@host ~]# sysctl -w net.ipv4.tcp_l3mdev_accept=1
net.ipv4.tcp_l3mdev_accept = 1
[root@host ~]# ip vrf exec vrf1234 nc 127.0.0.1 22 </dev/null | head -1
SSH-2.0-OpenSSH_8.0
[root@host ~]#
```

Figure 3. Output showing the effect of net.ipv4.tcp_l3mdev_accept

3.2.1. Comparison to Network Namespaces

An approach like this is not directly possible for network namespaces and would require that service instances be run in each network namespace for each service that should be universally available. Though it is viable to launch additional sshd instances in each network namespace, slurmd and many system services cannot currently handle that approach. This makes network namespaces difficult—but not impossible—to work with due to the need to set up complex Routing Policy Database (RPDB) rules both inside and outside of the namespace.

A mostly functional but brittle implementation of this project was created at one point using network namespaces, but it was extremely sensitive to differences in networking parameter defaults in various Linux kernel and distribution versions. The *ip rule* and *ip route* commands needed to set up the RPDB became quite complicated, as did some of the sysctl parameters that had to be adjusted. pam_slurm_adopt and its underlying infrastructure would have required substantial changes.

It's possible that a simpler version of this project could have been achieved with network namespaces if certain requirements were relaxed. However, substantial complexity would still exist due to the requirement that system services be available transparently over the network and on localhost inside of a namespace at the same IP address.

© 2021 The SANS Institute https://t.me/learningnets Author retains full rights.

During initial testing, VRFs were still required in the root namespace to attach the namespace's outer veth interface[16] to so that packets could be routed back into the right namespace from the system service; this would not have been a concern if unique IP addresses were assigned per user per interface rather than duplicating the IP addresses[17].

3.3. Shortcomings of Selected Technologies

VRFs do not prevent users from communicating with other users' programs over Unix sockets. Network namespaces would be necessary

VRFs and VXLAN, as with all non-InfiniBand-specific technologies, cannot isolate native InfiniBand traffic. PKEYs[18] are the only method for doing so with native InfiniBand traffic. This is important to note because InfiniBand networks are very common in HPC (Morgan, 2017). The lack of protection for native traffic may be detrimental in some cases, but attacks on InfiniBand are very rare[19]. VRFs and VXLAN can isolate IPoIB traffic, however.

4. Implementation

Each user who logs into a system is assigned to a unique VRF which contains a set of VXLAN interfaces. The VXLAN interfaces are configured with unique per-user VNIs. This creates an environment where users are unable to access each other's network resources, either locally or remotely.

---

[16] Network namespaces usually include one or more "veth" interfaces (Edge, 2014). veth interfaces are virtual interfaces that come in pairs, very similar in function to a physical network cable; a packet that is sent to a veth interface is sent out the paired veth interface. Typically, one veth interface is moved inside the namespace and one remains in the root namespace or is moved into a different namespace. The interfaces are then used for routing or as bridge members. This is one way to allow network communication to traverse namespace boundaries.

[17] This could be a paper topic by itself. See Appendix B for a brief explanation.

[18] Partition keys, also referred to as PKEYs or P_Keys, are an InfiniBand-specific technology used for partitioning communications on an InfiniBand fabric, very analogous to Ethernet VLANs (Muehlfeld et al., n.d.).

[19] The author did not conduct a thorough search for InfiniBand vulnerabilities since they are not the focus of this paper. However, documented vulnerabilities in InfiniBand are rare enough that it is difficult to find more than a few examples. The only two documented vulnerabilities the author found in a limited search were noted in a paper from 2005 (Lee et al.) and a paper from 2012 (Warren).

© 2021 The SANS Institute https://t.me/learningnets Author retains full rights.

The entire setup consists of a bash script, a Slurm SPANK plugin, and a line in a PAM configuration file. For lack of a better term, the author has chosen to refer to these setups as "User Virtual Networks", or "UVN" for short[20].

4.1. create_uvn script

A bash script, create_uvn, was written to assemble a per-user (i.e. per-UID) networking configuration when a user logs into a system over ssh or launches a job through Slurm. The script ensures that various sysctl and iptables settings are in place, reorders the RPDB to the correct ordering for VRFs ("Virtual Routing and Forwarding (VRF)", n.d.), creates a VRF per user, creates a VXLAN interface per user per "real" interface, and places a user's initial process in their VRF. It also handles locking to prevent concurrent execution attempts.

The script creates one VRF for the user. 127.0.0.1 and ::1 addresses[21] are added to the VRF so that those addresses continue to be accessible to the user; the loopback device "lo" only exists in the default VRF and would thus be inaccessible to the user due to reply packets not having a proper return path.

create_uvn then iterates through the available interfaces on the system, looking for IP addresses that are part of administrator-defined subnets of interest. An administrator would typically list all subnets of which nodes might be a member, whether on Ethernet, InfiniBand, or other networks. A VXLAN interface is created for the user on each discovered interface. This results in a per-user VXLAN interface for each "real" interface on the system.

For example, if em1 and ib0 both exist on a system and have IP addresses in the administrator-designated subnets, each user who logs in would have two VXLAN interfaces created for them that are bound to the "real" interfaces. The interfaces are

[20] Never put the author in charge of naming something.

[21] A "recent" kernel patch fixes a bug where ::1 could not be added to an l3mdev like 127.0.0.1 can (Shearman, 2018). create_uvn catches the failure to add ::1 and completely disables IPv6 if this occurs. IPv6 is still uncommon in HPC, so this was seen as an acceptable solution for a proof-of-concept script. The referenced patch has been included in recent kernels so this will soon become a non-issue.

© 2021 The SANS Institute https://t.me/learningnets Author retains full rights.

named with the user's UID as part of the interface name (e.g. "em1u1234" for UID 1234 on interface em1).

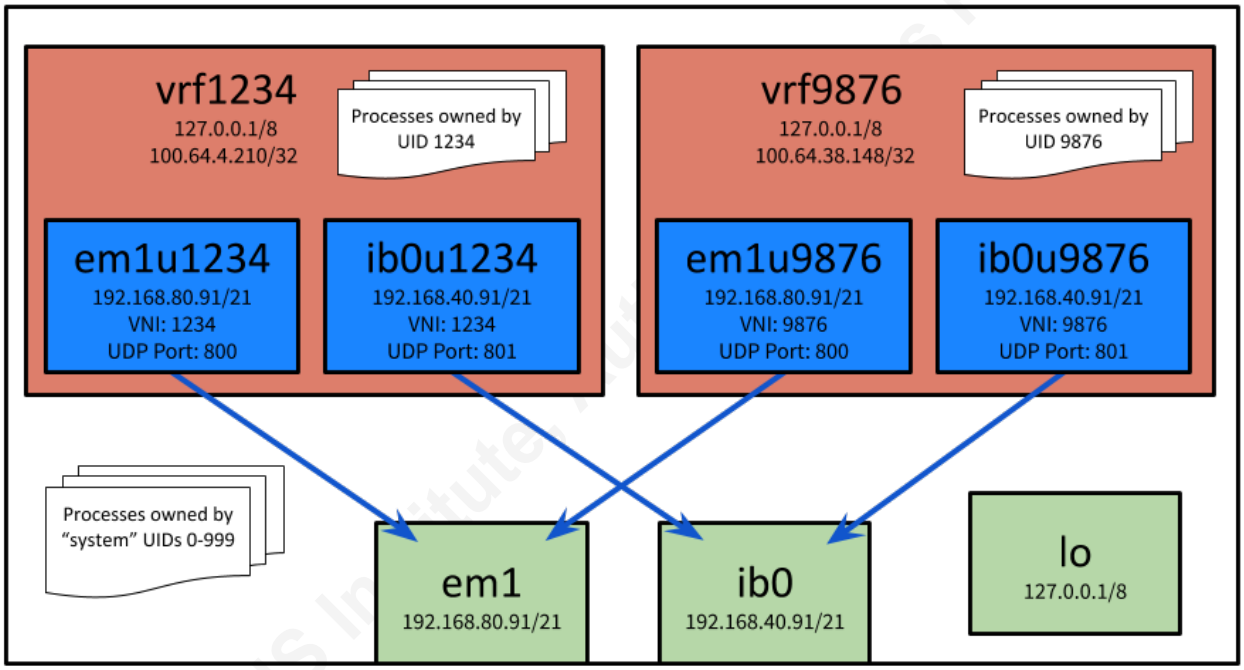

Figure 4. Example system configuration with two physical interfaces (green), a loopback interface (green), four VXLAN interfaces (blue), and two VRFs (red).

The VXLAN interface specifies a multicast IP that is common for all VXLAN interfaces. The user's UID is specified as the VNI, thus separating each user's traffic from other users. Each VXLAN interface would be created in an identical way, if not for an interesting implementation detail in Linux.

Multiple VXLAN interfaces in a Linux system cannot typically share the same VNI, even if the VXLAN interfaces are bound to different "real" interfaces (Bauvin, 2019). For example, multiple VXLAN interfaces with VNI 1234 cannot exist simultaneously even if one is attached to em1 with subnet 192.168.80.0/21 and the other is attached to ib0 with subnet 192.168.40.0/21.

A workaround is to specify a different UDP port for each VXLAN interface to communicate over (Derumier, 2018). The chosen UDP port must be common for VXLAN interfaces on all hosts that intend to use the same subnet. For example, 192.168.80.0/21 on VNI 1234 communicates using UDP port 800 while VNI 1234 on 192.168.40.0/21 communicates using UDP port 801. create_uvn, therefore, assigns each

https://t.me/learningnets
© 2021 The SANS Institute Author retains full rights.

administrator-defined subnet a unique UDP port for VXLAN to communicate on for all hosts in that subnet.

A user's VXLAN interfaces are attached to the user's VRF. Each VXLAN interface is configured to have the same IP address as the "real" interface it communicates over. For example, if em1 has an address of 192.168.80.91/21, em1u1234 will also have an address of 192.168.80.91/21. This is true for each user's em1 VXLAN interface. Due to the usage of VRFs, this duplication is permissible because each Layer 3 domain is isolated from the others.

The following code demonstrates the setup of a VXLAN interface and its attachment to the user's VRF, vrf$uid. $list_addresses and $list_interfaces are, respectively, arrays of IP addresses and the interfaces they reside on. $list_dstports is an array of UDP ports for VXLAN communication with a unique port per subnet, as described above.

```
# Loop through interfaces and create a matching vxlan interface
for ((i=0; i < ${#list_interfaces[@]}; i++))
do
        vxlan_interface="${list_interfaces[$i]}u$uid"

        #Create interface after checking if it already exists (e.g. multiple addresses on
the same interface)
        ip link show $vxlan_interface >/dev/null 2>&1 || \
        ip link add $vxlan_interface type vxlan group $MULTICAST_IP dev
${list_interfaces[$i]} ttl 5 id $uid dstport ${list_dstports[$i]}

        #Assign interface to VRF
        ip link set dev $vxlan_interface master vrf$uid

        #Add the address to the interface and bring up the interface
        ip addr add ${list_addresses[$i]} dev $vxlan_interface
        ip link set $vxlan_interface up
done
```

Figure 5. Creation of VXLAN interfaces in create_uvn

Each VRF has its own routing table ("Virtual Routing and Forwarding (VRF)", n.d.). The routing table is populated with locally-attached routes for the VXLAN interfaces. The routing table for each VRF is largely similar to the main routing table,

© 2021 The SANS Institute https://t.me/learningnets Author retains full rights.

except that the routes traverse the user-specific VXLAN interfaces in the VRF rather than the "real" interfaces.

create_uvn does not currently support IPv6. The author did not have a suitable test environment, so IPv6 support was omitted. Comments were left in the script to note where IPv6 equivalents should be added.

create_uvn is available at https://github.com/ryanbcox/uvn.

4.2. NAT for Off-Subnet Communication

Unfortunately, Network Address Translation (NAT) is currently required for off-subnet communication. Multiple VRFs cannot communicate on the same interface while sharing the same IP address (Red Hat, Inc., 2020). Outbound packets can successfully leave the same interface but return packets will not be delivered back to the sender if it is in a non-default VRF and has a duplicate IP address. Various unsuccessful attempts were made to overcome this limitation, such as using conntrack zones[22] (McHardy, 2010). More research is needed to determine if a solution to this limitation exists. NAT is therefore used to communicate off-subnet since it does not suffer from the same limitation.

In order to not compete with existing RFC 1918 address space, each VRF is assigned a host-scoped unique IP address from Carrier-Grade NAT (CGNAT) space in full compliance with RFC 6598. The address is assigned based on the user's UID to ensure uniqueness of addresses between users. The example in Figure 4 includes 100.64.0.0/10 CGNAT addresses on the VRF interfaces[23].

[22] conntrack zones seem to solve a similar problem for connection tracking. A numeric zone identifier can be applied to packets based on arbitrary criteria, typically the inbound or outbound device (McHardy, 2010). This makes it possible to use connection tracking without collisions while using the same IP address on multiple interfaces in the same network namespace. However, it wasn't clear that connection tracking is even the cause of the problem encountered with inbound packets in a non-default VRF. Further investigation indicated a more fundamental issue with using conntrack zones: the inbound interface in this scenario is the same for all packets received from off-subnet, thus inbound packets cannot be easily mapped to the appropriate zone.

[23] The addresses are specified as /32 because the kernel can route to them internally and they don't need to be available outside of the VRF.

© 2021 The SANS Institute https://t.me/learningnets Author retains full rights.

A default route in the user's VRF specifies its unique CGNAT address as the source address. Source NAT (SNAT) rules set up by iptables handle all packets from the CGNAT address space that are destined for a network that is not locally attached; the source addresses of the outbound packets are modified to be the "correct" IP address.

This is unlikely to cause usability problems for users. Most applications do not need to know their source address, so it is unlikely to matter. Additionally, root-owned and "system" processes that are not placed in a VRF will not be impacted. This means that file system traffic is unaffected since it is not placed in a VRF.

4.3. PAM and Slurm SPANK

The PAM configuration for sshd calls pam_exec.so and specifies the path to the create_uvn script. The script creates the user's network configuration then moves the sshd process into the user's VRF. Most applications that integrate with PAM could be configured in a similar way.

Though Slurm can integrate with PAM, PAM is not adequate for running the script on remote nodes. A problem occurs because the user's batch job can start on one node, but Slurm does not run PAM at job launch time on other nodes in the job. The user's UVN is thus set up by Slurm on the first node but is not set up on the other nodes. When a process in the batch job attempts to communicate with any other node that is part of the job, the ssh or Slurm traffic will be sent over the user's VXLAN interface, but the corresponding VXLAN interface will not exist on the destination node.

A Slurm SPANK plugin[24] was developed to set up UVN at job launch time on each node and to place later job steps[25] inside of that UVN. The plugin source code is also available at https://github.com/ryanbcox/uvn.

[24] SPANK plugins allow for administrators and developers to modify the Slurm batch job launch environment (SchedMD LLC, n.d.-b.).
[25] Slurm job steps are essentially sub-allocations within a job. One or more job steps exist for any job (SchedMD LLC, n.d.-a.).

© 2021 The SANS Institute Author retains full rights.
https://t.me/learningnets
© 2021 The SANS Institute, Author Retains Full Rights

4.4 Configuration

Configuration is as simple as installing the script and modifying two configuration files. For example, if the script is located at /usr/local/sbin/create_uvn, configuration might appear as follows:

```
/etc/authselect/postlogin:
        session  optional  pam_exec.so /usr/local/sbin/create_uvn

/etc/slurm/plugstack.conf.d/create_uvn.conf:
        optional /usr/local/spank/create_uvn.so
```

Figure 6. Configuration example

On Red Hat Enterprise Linux 8, /etc/pam.d/postlogin symlinks to /etc/authselect/postlogin and has existing configuration lines. Figure 6 only shows the additional line necessary to run create_uvn.

5. Testing

Two sets of automated tests were developed to assess the environment created by create_uvn with a configuration similar to the example specified in Section 4.4. The first set of tests validates the functionality. The second set of tests benchmarks the performance under a variety of conditions.

5.1. Functionality

Functionality was tested through a script. The tests ran through many conditions, including:

- User isolation works on localhost
- User isolation works between node
- Users can connect to resources off of the network where such connections are allowed
- Users can connect to a listening port owned by the same user on a different node
- Users can ping between identical VRFs on different nodes

Ryan Cox, ryan_cox@byu.edu

© 2021 The SANS Institute Author retains full rights.
https://t.me/learningnets
© 2021 The SANS Institute Author Retains Full Rights

- Users can use ssh to connect between nodes
- Users can use srun to launch batch job steps between nodes
- Users can use sbatch to launch batch jobs on the nodes
- Users can launch MPI programs across nodes

5.2. Performance

The following software packages were used to test performance.

5.2.1. iperf3

The iperf3 benchmark was used to perform bandwidth tests between each node in the group. Tests were run for 120 seconds each with 10 seconds of warmup time (*-O 10*) on an otherwise idle system. iperf3 instances were launched with an increasing number of simultaneous instances, up to eight in total.

iperf3 is very useful because it can easily perform CPU pinning of both the client and server processes, even remotely. The tests pinned each iperf3 instance to its own CPU core and each core was chosen based on physical locality of the core and NIC. Since each node contains two processors, this was a very important issue to address. Quick, informal checks by the author determined that bandwidth could decrease by more than 15% if CPU cores were chosen that were not physically located on the CPU to which the NIC was attached.

5.2.2. Intel MPI Benchmarks IMB-P2P

Since MPI is so widely used in HPC, it was essential to test several multi-node communication patterns with MPI. The IMB-P2P tests from Intel MPI Benchmarks 2019 update 6 were chosen for this test. A more extensive test with additional components would have been preferred, but the tests mentioned in this section already resulted in hundreds of graphs and tables to compare.

IMB-P2P tests were performed in two configurations. The first configuration used a single processor core per node. The second configuration used all processor cores on each node.

© 2021 The SANS Institute https://t.me/learningnets Author retains full rights.

5.2.3. High Performance Linpack

High Performance Linpack (HPL) is a widely adopted benchmark in HPC that is often used to compare systems. HPL "is a software package that solves a (random) dense linear system in double precision (64 bits) arithmetic on distributed-memory computers" (Petitet et al., 2018). HPL was run between nodes and used the same Intel MPI version and compilers as the Intel MPI Benchmarks. Tests were run across nodes using all processor cores. An attempt was made to configure HPL for maximum performance, but the intent was to compare UVN's performance impact to the performance of an unmodified system; the benchmark scores were likely slightly lower than they could have been.

Performance tests were conducted with four sets of nodes. Only two nodes with 100 Gb/s InfiniBand[26] were available for testing. The other three sets of nodes each have 56 Gb/s InfiniBand[27] and are identical in hardware configuration to each other; more of these nodes were available for testing so larger-scale tests were run there. The fastest Ethernet interfaces tested were 25 Gb/s.

All Ethernet subnets were configured for an MTU of 9000. For IPoIB, MTUs were set at either 2044 (Datagram Mode) or 65520 (Connected Mode)[28]. Some unexpected results occurred with newer Mellanox InfiniBand cards and drivers; there were interesting performance differences between in-kernel drivers and Mellanox-provided drivers on those cards. Each had benefits and drawbacks, so each was tested.

5.3. Node configurations

Nodes were chosen such that their network connections were made at the rated network card speed to a non-blocking switch, whether Ethernet or InfiniBand, and the nodes were only one hop away from each other. In other words, each node in a group

[26] The 100 Gb/s InfiniBand cards and cables are 4X EDR. The cards are Mellanox ConnectX-5.
[27] The 56 Gb/s InfiniBand cards and cables are 4X FDR. The cards are Mellanox ConnectX-3.
[28] IPoIB can use one of two modes that have different advantages and disadvantages (Kashyap, 2006). A big advantage of Connected Mode is that the maximum MTU size is 65520 rather than the typical 2044 MTU for Datagram Mode (Lawrence Berkeley National Laboratory, n.d.).

© 2021 The SANS Institute https://t.me/learningnets Author retains full rights.

was physically plugged into the same switches as the other nodes in the group. Port channels were not used during testing.

Due to the availability of resources, no effort could be made to prevent unrelated traffic from traversing the same switches and interfering with the tests. The switches in use are rated to function at line-rate with all ports fully loaded, so any resource contention should have been negligible. All nodes were configured in the BIOS and operating system for maximum performance.

Nodes were tested in the following groups:

- *Group A*: Two nodes, each with:
  - dual Intel Xeon Gold 6230 2.10 GHz processors with 20 cores each, totaling 40 cores
  - 768 GiB RAM
  - dual 25 Gb/s QLogic FastLinQ QL41000 Ethernet card, each connected to a separate switch
  - dual Mellanox MT27800 100 Gb/s EDR InfiniBand cards, each connected to a separate switch
- *Group B*: Up to sixteen nodes, each with:
  - dual Intel Xeon CPU E5-2680v4 2.40GHz processors with 14 cores each, totaling 28 cores
  - 128 GiB RAM
  - 10 Gb/s Intel 82599ES Ethernet card
  - Mellanox MT27500 56 Gb/s FDR InfiniBand card

5.3.1. Two-node tests

A pair of nodes from Group A and a pair from Group B were used for two-node testing. All tests were performed across all available network cards, both Ethernet and InfiniBand. Tests over InfiniBand were conducted using IPoIB. For Intel MPI Benchmarks and HPL, an additional test was performed in each node group where MPI

© 2021 The SANS Institute https://t.me/learningnets Author retains full rights.

was allowed to use its default discovery method. In each case, MPI chose the fastest InfiniBand interface and used native InfiniBand rather than IPoIB.

For each piece of software listed in the section above, the tests were performed inside and outside a UVN setup created by create_uvn. The VRF and VXLAN interfaces already existed during all tests, whether they were used in that particular test or not. The systems were not rebooted or otherwise altered to remove those additional interfaces in between tests.

5.3.2. Eight- and sixteen-node tests

The eight- and sixteen-node tests omitted the iperf3 bandwidth tests since those tests are only applicable between two nodes. The IMB-P2P and HPL benchmarks were run in a similar manner to the two-node benchmarks.

6. Analysis

6.1. Functionality

All tests passed successfully on multiple sets of nodes, whether IPoIB was in use or not. The solution works as designed; users were isolated from each other over the network and on localhost.

6.2. Performance

Performance testing showed negligible performance impact in some areas and greater impacts in others. A large number of test types were run. Unfortunately, there were so many data points to examine that it's difficult to succinctly compare the performance of this solution. The raw data is available at https://github.com/ryanbcox/uvn.

6.2.1. iperf3

6.2.1.1. iperf3 on 25 Gb/s Ethernet

iperf3 showed minimal performance degradation of about 1-2% for UVN on 25 Gb/s Ethernet, depending on how many concurrent iperf3 instances were run.

© 2021 The SANS Institute https://t.me/learningnets Author retains full rights.

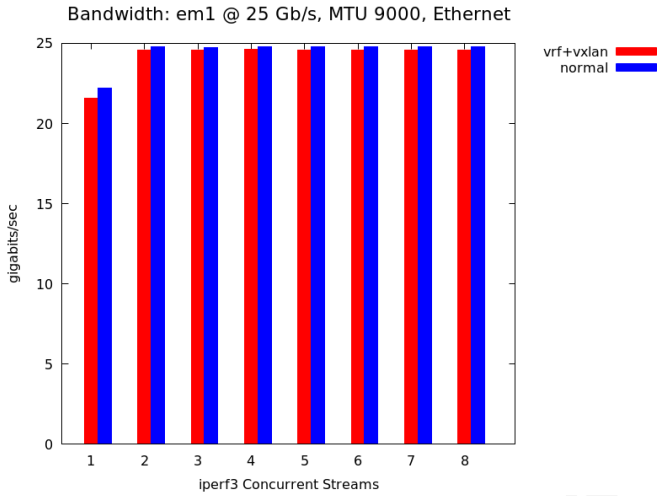

Figure 7. 2-node Group A. iperf3 on 25 Gb/s Ethernet

6.2.1.2. iperf3 on InfiniBand

IP over InfiniBand showed dramatic differences depending on if official Mellanox drivers were used or not. The Mellanox drivers exhibited terrible scaling for more than two iperf3 instances, regardless of whether VRF and VXLAN were in use or not[29].

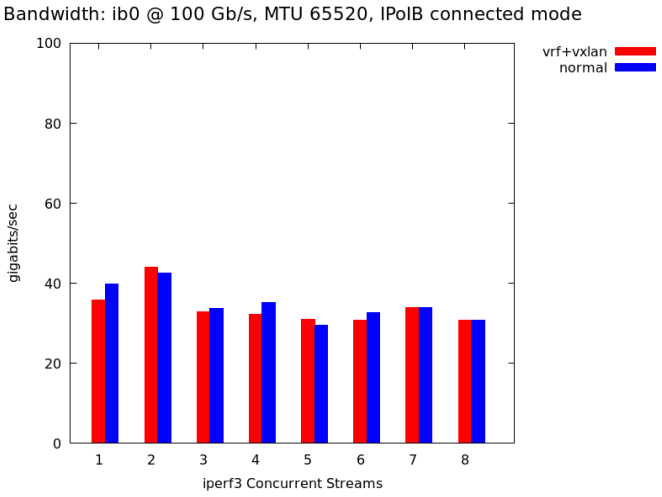

Figure 8. 2-node Group A. 100 Gb/s IPoIB with Mellanox drivers

[29] A support ticket was opened with the vendor but the issue was never resolved. The author tested 100 Gb/s InfiniBand with and without drivers from Mellanox OFED.

https://t.me/learningnets
© 2021 The SANS Institute Author retains full rights.

One and two streams achieved good performance but beyond that the results were poor. The performance dropped off some after that but the UVN and non-UVN results were close; sometimes UVN outperformed the non-UVN setup for an unknown reason.

These issues are unfortunate since the Mellanox drivers are required for 100 Gb/s cards to use Connected Mode and thus the 65520 byte MTU. Datagram Mode in the test environment could only use a 2044 byte MTU for the IPoIB interface.

Both Connected and Datagram Modes were tested on 100 Gb/s IPoIB. Due to time limitations, testing on 56 Gb/s IPoIB was limited to Connected Mode.

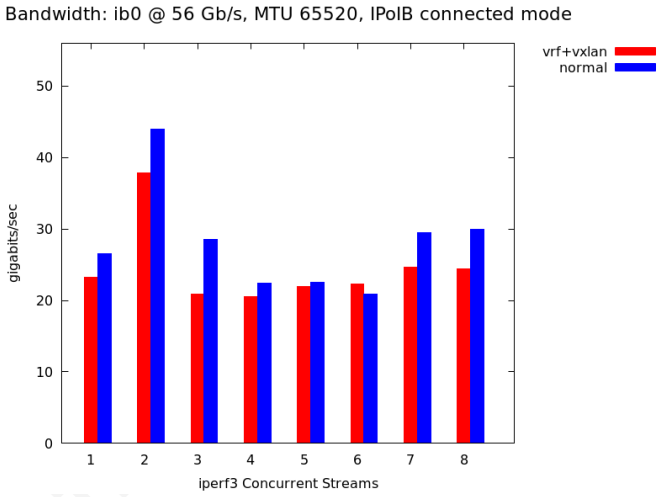

Figure 9. 2-node Group B. iperf3 on 56 Gb/s InfiniBand with Mellanox drivers

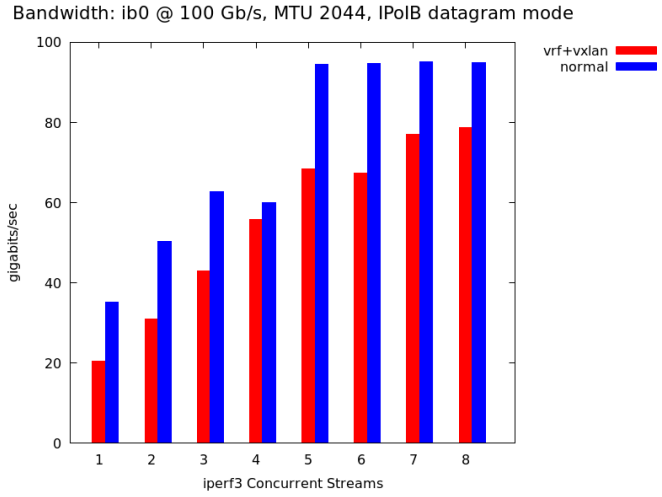

Figure 10. 2-node Group A. 100 Gb/s IPoIB with stock IB drivers

Ryan Cox, ryan_cox@byu.edu

https://t.me/learningnets
© 2021 The SANS Institute Author retains full rights.

Performance degradation for UVN was severe when using UVN with Datagram Mode on stock drivers, but it did scale fairly well. It is unclear if the performance difference is the result of the small MTU or other factors. The VXLAN MTU is 1994 bytes, an almost 2.5% decrease in MTU size for an IPoIB MTU of 2044 bytes. As message size decreases, the number of packets needed to achieve a given bandwidth target increases. CPU usage can become a bigger issue as more packets have to be processed by the CPU (Lawrence Berkeley National Laboratory, n.d.).

6.2.2. IMB-P2P

Hundreds of graphs and tables were generated with IMB-P2P results due to the large number of tests that were run. A sampling of results is presented here to showcase good, bad, and interesting results for UVN. All graphs are available at https://github.com/ryanbcox/uvn.

IMB-P2P results did not paint a clear picture of the performance impact of UVN. Differing link speeds, MTUs, and Layer 2 technology resulted in different performance profiles for UVN compared to a non-UVN setup. For example, the 2-node IMB PingPong bandwidth test with 40 cores per node[30] on 100 Gb/s InfiniBand with IPoIB Connected Mode showed good overall performance for UVN (see Figure 11), but that same test on 25 Gb/s Ethernet showed a small but widening advantage for non-UVN until the 8192-byte message size (see Figure 13). After that point, UVN performance plummeted.

The different performance profile does not have a readily apparent explanation. UVN performed well on 100 Gb/s InfiniBand but terribly on 25 Gb/s Ethernet. InfiniBand has much lower latency than Ethernet, but any advantages should apply to both UVN and non-UVN configurations. MTU sizes were very different between the InfiniBand and Ethernet tests, but the extreme performance degradation on 25 Gb/s Ethernet does not seem attributable to the higher overhead of the smaller Ethernet MTU.

[30] Labelled "ppn=40" (meaning 40 processors per node) on graphs using common, though slightly incorrect, traditional scheduler and MPI terminology where "processor" actually refers to a processor core.

© 2021 The SANS Institute Author retains full rights.
https://t.me/learningnets
© 2021 The SANS Institute Author Retains Full Rights

However, an MTU difference likely played a major role in the differences between Datagram Mode with the stock InfiniBand drivers and Connected Mode with Mellanox drivers (see Figures 11 and 12). This results in a 2044-byte MTU for IPoIB and a 1994-byte MTU for VXLAN. UVN's performance in Datagram Mode shows some degradation at a certain range of message sizes but it usually significantly outperforms both UVN and non-UVN setups at the same message size in Connected Mode.

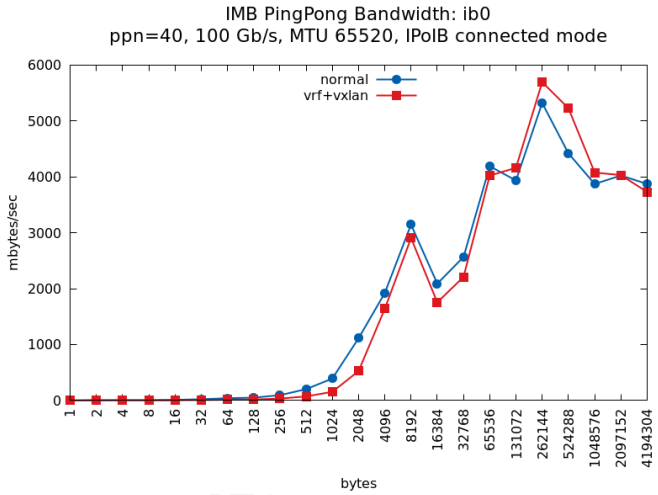

Figure 11. 2-node Group A. 100 Gb/s IPoIB with Mellanox IB drivers

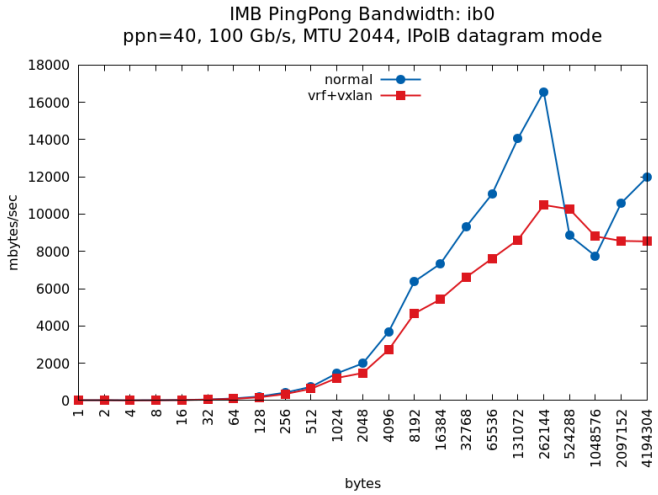

Figure 12. 2-node Group A. 100 Gb/s IPoIB with Mellanox IB drivers

https://t.me/learningnets
© 2021 The SANS Institute Author retains full rights.

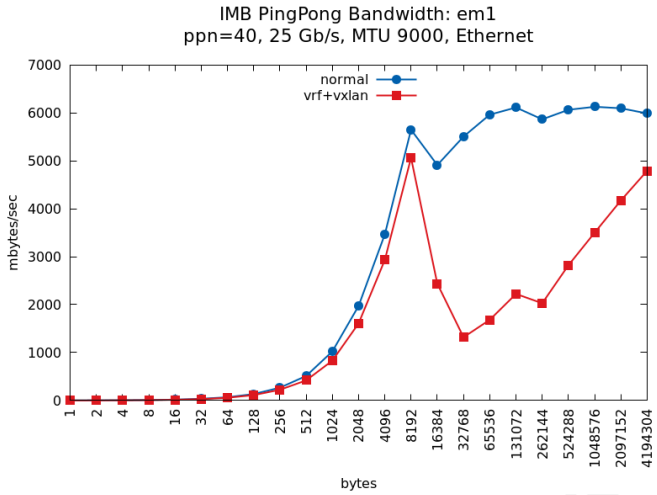

Figure 13. 2-node Group A. 25 Gb/s Ethernet

While some of these graphs are not flattering for UVN, many of the tests were very similar to the latency graph in Figure 14 and showed only minor impacts when using UVN. A significant number of graphs showed similar performance between UVN and non-UVN setups.

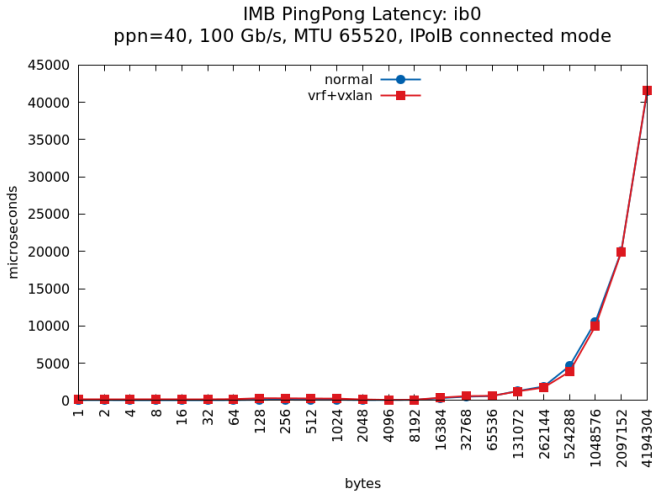

Figure 14. 2-node Group A. 100 Gb/s IPoIB Connected Mode

6.2.3. HPL

HPL results showed some performance impact depending on the node count. The 16-node tests showed the most performance degradation compared to the smaller tests, but the results seem reasonable for a latency-sensitive application such as HPL. It is

https://t.me/learningnets
© 2021 The SANS Institute Author retains full rights.

expected that the performance gap would widen with higher node counts, though more testing is needed.

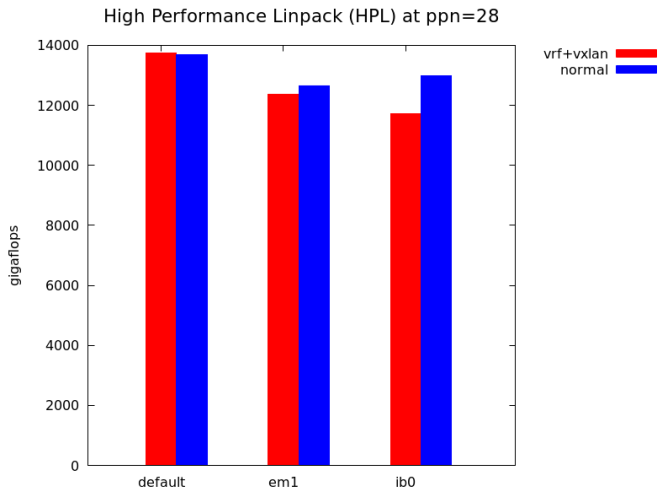

Figure 15. 16-node Group B. em1: 25 Gb/s. ib0: 56 Gb/s Connected Mode

7. Conclusions and Future Research

The solution works for its intended purpose and is easy to administer. It integrates seamlessly with ssh and the batch job scheduler Slurm. No modifications to user code, configurations, or workflows were required. MPI works flawlessly.

Testing confirmed the functionality of create_uvn and the environment it creates; it works perfectly. Users were completely segregated from each other from a network perspective on both local and remote hosts, aside from native InfiniBand traffic and Unix sockets. System services continue to be available without modification.

The solution is easy to install. Only two files are necessary: the bash script and a simple Slurm SPANK plugin. pam_exec.so is used to call the script for ssh connections.

VRFs are superior to network namespaces for this use case. VXLAN worked perfectly in this project, though better encapsulation technologies may exist.

Performance tests were promising, though far from definitive. Many real workloads would experience a negligible impact at most, even across eight or sixteen nodes. However, a subset of MPI benchmarks showed severe performance degradation

© 2021 The SANS Institute https://t.me/learningnets Author retains full rights.

when run across sixteen nodes; it is unclear what triggers the degradation of those tests' communication patterns. The performance tests could be considered a stress test of the solution using various communication patterns, something that is hopefully not done by real code in production.

Multi-node HPL runs further demonstrated that some MPI programs are only negligibly impacted, though even that impact may be attributable to jitter. Further performance testing is needed using actual HPC applications rather than benchmark tools.

The combination of VRFs and VXLAN is effective and simple to deploy. It is a good addition to a defense-in-depth strategy, assuming that the performance characteristics are acceptable to the HPC site.

© 2021 The SANS Institute, Author Retains Full Rights

https://t.me/learningnets
© 2021 The SANS Institute Author retains full rights.

References

Ahern, D. (2017, March 18). *VRF for Linux — a contribution to the Linux Kernel*. Cumulus Networks engineering blog. https://cumulusnetworks.com/blog/vrf-for-linux/

Balaji, P., Buntinas, D., Goodell, D., Gropp, W., Krishna, J., Lusk, E., & Thakur, R. (2010, September). PMI: A Scalable Parallel Process-Management Interface for Extreme-Scale Systems. *EuroMPI 2010*. 10.1007/978-3-642-15646-5_4

Bauvin, A. (2019, October 09). *Re: ip doesn't handle vxlan id and group correctly*. Linux Kernel Mailing List. https://lore.kernel.org/netdev/BD89B7A5-AEA1-4DBD-BF4E-7330558162EF@online.net/t/#md1ca85ef7a59330055886ea648e6dab295f37bdf

Cox, R. B. (2015, November 16). *pam_slurm_adopt*. http://tech.ryancox.net/2015/11/pamslurmadopt.html

Cox, R. B. (2020, August 06). *Severe performance regression in "net: macsec: preserve ingress frame ordering"*. Linux Kernel Mailing List. https://lore.kernel.org/netdev/1b0cec71-d084-8153-2ba4-72ce71abeb65@byu.edu/T/

Dell EMC. (n.d.). *VLAN Stacking*. Dell EMC Networking OS Configuration Guide for the S5048F–ON System 9.14.2.6. Retrieved September 16, 2020, from https://www.dell.com/support/manuals/en-us/networking-

© 2021 The SANS Institute, Author Retains Full Rights
https://t.me/learningnets
© 2021 The SANS Institute Author retains full rights.

s5048f-on/s5048f-on-9.14.2.6-config-pub/vlan-stacking?guid=guid-9791044c-5b88-48de-b4db-3b5b75719717&lang=en-us

Derumier, A. (2018, December 13). *Re: [pve-devel] [PATCH pve-docs 0/1] vxlan l3 routing*. https://www.mail-archive.com/pve-devel@pve.proxmox.com/msg31244.html

Edge, J. (2014, January 22). *Namespaces in operation, part 7: Network namespaces*. LWN.net. https://lwn.net/Articles/580893/

Electric Sheep Fencing LLC & Rubicon Communications LLC. (n.d.). *pfSense QinQ Configuration*. Netgate Docs. Retrieved September 29, 2020, from https://docs.netgate.com/pfsense/en/latest/interfaces/qinq.html

IEEE. (2006, August 18). *IEEE Standard for Local and Metropolitan Area Networks: Media Access Control (MAC) Security*. https://ieeexplore.ieee.org/servlet/opac?punumber=11085

Kashyap, V. (2006, December). *IP over InfiniBand: Connected Mode*. https://tools.ietf.org/html/rfc4755

Lawrence Berkeley National Laboratory. (n.d.). *MTU Issues*. ESnet Fasterdata Knowledge Base. Retrieved November 11, 2020, from https://fasterdata.es.net/network-tuning/mtu-issues/

Lee, M., Kim, E. J., & Yousif, M. (2005, April). *Security Enhancement in InfiniBand Architecture*. https://people.engr.tamu.edu/ejkim/HPC_WEB/docs/ipdps.pdf

Litvak, L. (2017, November 24). *ip-link(8)*. https://manpages.debian.org/stretch/iproute2/ip-link.8.en.html

© 2021 The SANS Institute https://t.me/learningnets Author retains full rights.

Liu, H. (2018, October 22). *Introduction to Linux interfaces for virtual networking*. Red Hat Developer. https://developers.redhat.com/blog/2018/10/22/introduction-to-linux-interfaces-for-virtual-networking/#vxlan

Lunn, A. (2020, August 12). *Re: Severe performance regression in "net: macsec: preserve ingress frame ordering"*. Linux Kernel Mailing List. https://lore.kernel.org/netdev/20200812124201.GF2154440@lunn.ch/

Mahalingam, M., Dutt, D., Duda, K., Agarwal, P., Kreeger, L., Sridhar, T., Bursell, M., & Wright, C. (2014, August). *Virtual eXtensible Local Area Network (VXLAN): A Framework for Overlaying Virtualized Layer 2 Networks over Layer 3 Networks*. https://tools.ietf.org/html/rfc7348

McHardy, P. (2010, January 14). *RFC: netfilter: nf_conntrack: add support for "conntrack zones"*. netfilter-devel Mailing List. https://marc.info/?l=netfilter-devel&m=126347794223371

Morgan, A. G., & Kukuk, T. (2010, August 31). *The Linux-PAM System Administrators' Guide*. http://www.linux-pam.org/Linux-PAM-html/sag-introduction.html

Morgan, T. P. (2017, June 30). *InfiniBand And Proprietary Networks Still Rule Real HPC*. The Next Platform. https://www.nextplatform.com/2017/06/30/infiniband-proprietary-networks-still-rule-real-hpc/

Muehlfeld, M., Jahoda, M., Heves, J., Wadeley, S., & Huffman, C. (n.d.). *13.8. Configuring IPoIB*. Red Hat Enterprise Linux 7 Networking Guide.

© 2021 The SANS Institute https://t.me/learningnets Author retains full rights.

Retrieved November 10, 2020, from https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/7/html/networking_guide/sec-configuring_ipoib

The Open MPI Project. (2019, May 20). *FAQ: Running MPI jobs*. https://www.open-mpi.org/faq/?category=running#diagnose-multi-host-problems

Petitet, A., Whaley, R. C., Dongarra, J., & Cleary, A. (2018, December 2). *HPL - A Portable Implementation of the High-Performance Linpack Benchmark for Distributed-Memory Computers*. https://www.netlib.org/benchmark/hpl/

Project Jupyter. (n.d.). *Project Jupyter*. Retrieved December 2, 2020, from https://jupyter.org/

Red Hat, Inc. (2020, September 25). *Chapter 33. Configuring virtual routing and forwarding (VRF)*. Red Hat Enterprise Linux 8 Configuring and managing networking. https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/8/html/configuring_and_managing_networking/configuring-virtual-routing-and-forwarding-vrf_configuring-and-managing-networking

Rekhter, Y., Moskowitz, B., Karrenberg, D., de Groot, G. J., & Lear, E. (1996, February). *Address Allocation for Private Internets*. https://tools.ietf.org/html/rfc1918

© 2021 The SANS Institute https://t.me/learningnets Author retains full rights.

Sandve, G. K., Nekrutenko, A., Taylor, J., & Hovig, E. (2013, October 24). Ten Simple Rules for Reproducible Computational Research. *PLOS Computational Biology*, *9*(10), 1-4. https://doi.org/10.1371/journal.pcbi.1003285

SchedMD LLC. (n.d.-a.). *Quick Start User Guide*. Retrieved November 14, 2020, from https://slurm.schedmd.com/quickstart.html

SchedMD LLC. (n.d.-b.). *SPANK*. Slurm Workload Manager. Retrieved August 31, 2020, from https://slurm.schedmd.com/spank.html

Schmaus, B. (2017, June 22). *What is GENEVE?* Red Hat Blog. https://www.redhat.com/en/blog/what-geneve

Shearman, R. (2018, September 19). *[net-next] ipv6: Allow the l3mdev to be a loopback*. Patchwork. https://patchwork.ozlabs.org/project/netdev/patch/20180919125653.15913-1-mmanning@vyatta.att-mail.com/

Trader, T. (2019, September 11). *Univa Brings Cloud Automation to Slurm Users with Navops Launch 2.0*. HPCwire. https://www.hpcwire.com/2019/09/11/univa-brings-cloud-automation-to-slurm-users-with-navops-2-0-launch/

Virtual Routing and Forwarding (VRF). (n.d.). kernel.org. Retrieved October 28, 2020, from https://www.kernel.org/doc/Documentation/networking/vrf.txt

Warren, A. (2012, October). *InfiniBand Fabric and Userland Attacks*. https://www.sans.org/reading-room/whitepapers/incident/infiniband-fabric-userland-attacks-34012

© 2021 The SANS Institute https://t.me/learningnets Author retains full rights.

Weil, J., Kuarsingh, V., Donley, C., Liljenstolpe, C., & Azinger, M. (2012, April). *IANA-Reserved IPv4 Prefix for Shared Address Space*. https://tools.ietf.org/html/rfc6598

Ryan Cox, ryan_cox@byu.edu

© 2021 The SANS Institute, Author Retains Full Rights
https://t.me/learningnets
© 2021 The SANS Institute Author retains full rights.

Appendix A: Comparison of VXLAN to other technologies

A.1. Comparison of VXLAN to MACsec

MACsec is a viable technology for this project to use on Ethernet networks; the author initially explored it as a partial solution. The protocol has some great properties, such as working seamlessly at Layer 2 on Ethernet with no network configuration changes (IEEE, 2006).

Unfortunately, the Linux MACsec implementation contains some performance issues and regressions that were discovered and reported during initial testing for this paper (Cox, 2020). It was discovered that Linux MACsec on typical server hardware is significantly slower with encryption *disabled* than it is with encryption *enabled*. Likely due to this problem, MACsec on the author's test hardware could not fill a 25 Gb/s network link with or without encryption. An unrelated severe performance regression, introduced in Linux 5.7, was also identified and reported. The issue was acknowledged but there is currently no progress towards a resolution (Cox, 2020).

MACsec does not benefit from generic offloading technologies since it is an Ethernet protocol, thus some performance overhead exists compared to UDP-based encapsulation protocols. Though many NICs now support MACsec hardware offloading, Linux support for the offloading is still in its infancy (Lunn, 2020). MACsec appears to be a great technology for Ethernet but is still rough around the edges on Linux, has some performance overhead, and has very little documentation. It cannot work on InfiniBand.

A.2. VLANs and Stacked VLANs

Users could each be assigned their own VLAN ID. VLANs are a well-known, proven technology that might also be viable if not for a few downsides. One limiting factor for VLANs is that only 4,094 identifiers are available. This may not be a concern if an HPC site has 4,094 or fewer users, though a UID to VLAN mapping mechanism would likely be needed if there are gaps in the assigned UIDs or in existing VLAN

© 2021 The SANS Institute https://t.me/learningnets Author retains full rights.

numbering. Many HPC sites already use VLANs, thus stacked VLANs[31] would likely be required instead.

Depending on the implementation, many MAC addresses would be seen by intermediate switches as the number of users is multiplied by the number of nodes on the network[32]. If the number of MAC addresses exceeds network switches' available table space, this could result in significant performance degradation (Mahalingam et al., 2014). VXLAN does not have this problem since intermediate switches only examine the UDP packet's outer headers.

Though VLANs do not work directly on InfiniBand, InfiniBand PKEYs are a very close approximation of VLANs (Muehlfeld et al., n.d.). PKEYs even have the benefit of protecting native InfiniBand traffic that doesn't use IP.

A.3. Comparison of VXLAN to a Subnet Per User

A similar approach to a VLAN per user is to create a subnet per user per "real" subnet. The per-user subnet would have its IP addresses mapped to corresponding IP addresses in the existing "real" subnet. For example, 192.168.18.135/24 could be mapped to 100.67.18.135/24, where a certain user is assigned the 100.67.18.0/24 subnet as their own. When a user communicates across the network to a given node, they do so using the IP address for that node that is mapped into the user's subnet. Collisions with existing RFC 1918 address space (Rekhter et al., 1996) would likely be a deployment challenge, so it would be advantageous to utilize RFC 6598 Carrier-Grade NAT (CGNAT) address space (Weil et al., 2012).

Unfortunately, this solution would violate some of the author's constraints on not impacting usability. Having a subnet per user necessarily changes the IP addressing to no longer match DNS. If consistency with DNS is desired, as it is for the author, various workarounds would have to be employed to present a different view of DNS to each

[31] IEEE 802.1ad is often referred to as "stacked VLANs" or "QinQ" (Electric Sheep Fencing LLC & Rubicon Communications LLC, n.d.).

[32] Some network vendors caution against using non-unique MAC addresses in inner VLANs when using stacked VLANs (Dell EMC, n.d.). If this advice is followed, each user would need a unique MAC address on each interface on each node.

© 2021 The SANS Institute https://t.me/learningnets Author retains full rights.

user's processes. This is likely viable but the author did not test this approach; solutions would likely be specific to individual HPC sites. Performance might be better than VXLAN since no encapsulation is involved.

It is unclear if Slurm or MPI would be impacted by this configuration. Similar to the VLAN or stacked VLAN approach, it's possible that a particular implementation might result in too many MAC addresses being stored by switches.

© 2021 The SANS Institute, Author Retains Full Rights

https://t.me/learningnets
© 2021 The SANS Institute Author retains full rights.

Appendix B: Additional Information for Footnote 17

Based on the author's experience, routing is complicated when duplicate IP addresses and network namespaces are used. If traffic destined for a "system" service, such as slurmd, originated from inside a user's namespace or from that user on a remote system, the traffic has to be routed through the namespace and over the veth pair to the root namespace.

In this example, slurmd would try to reply back to the source IP address but would not route the packet back through the correct interface since there is necessarily a route in the global namespace pointing to that source IP address. This can be worked around by placing the user's root namespace veth interface inside of a user-specific VRF in the root namespace and by enabling the net.ipv4.tcp_l3mdev_accept family of sysctl knobs.

Since the traffic comes in through a VRF, the traffic is transparently sent back through that same VRF, with there being a route in the VRF that sends the return traffic through the veth interface into the namespace. Several sysctl parameters, routes, and rules have to be set inside each network namespace to permit the incoming traffic to pass through without being dropped since there would appear to be "martian" packets entering from the outside.

Ryan Cox, ryan_cox@byu.edu

© 2021 The SANS Institute https://t.me/learningnets Author retains full rights.