vMX Lightweight 4over6 Virtual Network Function

Juniper and Snabb in a Docker Container

Table of Contents

List of Figures

Executive Summary

Juniper Networks® vMX virtual router, packaged with the open-source fast packet networking engine Snabb in a Docker Container, delivers a highly scalable and high-performance virtual network function (VNF) to terminate millions of IPv4-in-IPv6 tunnels as Address Family Transition Routers (AFTRs), per RFC 7596.

Introduction

As service providers accelerate the migration of their core networks to IPv6, they need to ensure uninterrupted access and service continuity for all existing IPv4 users. This white paper describes the IPv6 transition challenges service providers are facing, and how Lightweight 4over6, as presented by the IETF RFC 7596, can help. Additionally, this white paper offers a detailed technical explanation of the main building blocks leading to a scalable and high-performance VNF, which is required at the service provider edge.

Packaged and run within a Docker Container, the Juniper Networks vMX virtual router integrates and fully manages an open-source, third-party, packet processing application (Snabb) to deliver a high-performance and scalable AFTR, and connect the Intel 82599-based 10GbE interfaces to the vMX virtual network interfaces. The IETF YANG Data Model for IPv4-in-IPv6 softwire is integrated with the vMX to configure and manage Snabb.

From a network integration perspective, the vMX Lightweight 4over6 VNF is a fully functional virtual routing solution, transparently augmented by a special purpose third-party "micro VNF" in the data path. In addition to AFTR, the vMX also supports multiple network protocol functions, including Address Resolution Protocol (ARP), IPv6 Neighbor Discovery Protocol (NDP), Open Shortest Path First (OSPF), Bidirectional Forwarding Direction (BFD), Border Gateway Protocol (BGP), Simple Network Management Protocol (SNMP), system logging, Network Configuration Protocol (NETCONF), and more for a truly carrier-grade virtual solution.

Each 10GbE interface is capable of terminating millions of tunnels. The stateless nature of the AFTR tunnel termination function allows a single VNF to not only scale up to multiple 10GbE interfaces, but also to scale out by sharing network traffic across several VNFs.

The IPv6 Transition Problem

Public IPv4 address exhaustion[1] is forcing service providers not only to accelerate IPv6 adoption, but also to share public IPv4 addresses among their subscribers. A simple dual-stack approach, as shown in Figure 1, consumes one public IPv4 address per residential subscriber. The most common solution today uses a carrier-grade NAT (CGNAT) (shown on the right side of Figure 1), also known as a large-scale Network Address Translation (NAT), which sits between the customer premises equipment (CPE) and the core network. Juniper offers such a solution today based on the MX Series 3D Universal Edge Routers (see the Day One: Deploying CGNAT on the MX Series guide for more details). The CGNAT router provides stateful Network Address Port Translation (NAPT) for all CPE devices to a small range of public IPv4 addresses. CGNAT requires the use of private IPv4 addresses assigned to individual CPE devices, making this approach incredibly complex for offering inbound services over IPv4, not to mention the numerous other challenges related to resiliency, scalability, and cost.

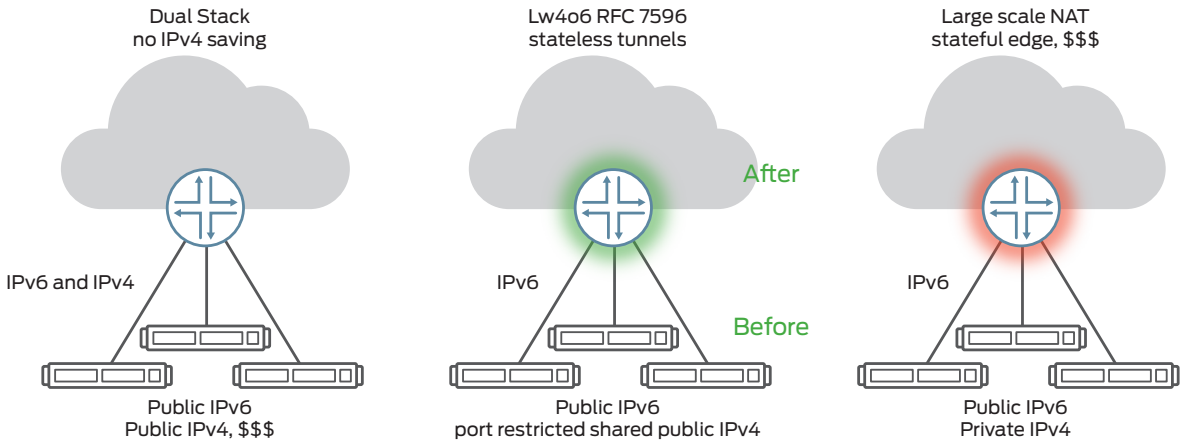

Figure 1: IPv4-to-IPv6 transition problem

[1] https://en.wikipedia.org/wiki/IPv4_address_exhaustion

Lightweight 4over6, defined in RFC 7596 and shown in the middle of Figure 1, delegates the stateful network address and port translation functionality to the CPE, while transporting the IPv4 traffic within an IPv6 stateless tunnel. The transport mechanism is derived from Dual-Stack Lite (DS-Lite) as defined by IETF RFC 6333. This removes the need for any address translation and stateful tunnel termination at the provider edge. In order to delegate the NAPT function and make IPv4 address sharing possible, port-restricted IPv4 addresses are allocated to the CPEs.

A single public IPv4 address can be shared by multiple subscribers, each being assigned a unique port range of 1,024 TCP/UDP ports. The port range from 0 to 1,023 is reserved for incoming services like SSH or HTTP, and can be mapped and optionally charged to select subscribers.

Table 1 depicts a typical binding table.

Table 1: Example Binding Table

Subscriber Public IPv6	Shared Public IPv4	UDP/TCP Port Range
2001:db8:1:1::1	193.5.1.1	1024 - 2047
2001:db8:1:1::2	193.5.1.1	2048 - 3071
2001:db8:1:1::3	193.5.1.1	3072 - 4095
. . .	. . .	. . .
2001:db8:1:1::63	193.5.1.1	64512 - 65535
2001:db8:1:1::64	193.5.1.2	1024 - 2047

vMX lwAFTR Docker Container

The vMX virtual router is launched within a Docker Container and connected to physical 10GbE interfaces via Snabb-based user mode network card drivers. All required applications, interface drivers, libraries, and daemons required to run the vMX virtual forwarding plane (VFP) and virtual control plane (VCP), including Qemu and KVM, are packaged in the Docker Container image. The vMX image itself is not distributed within the Docker Container image; it is added at startup, together with the configuration and license files.

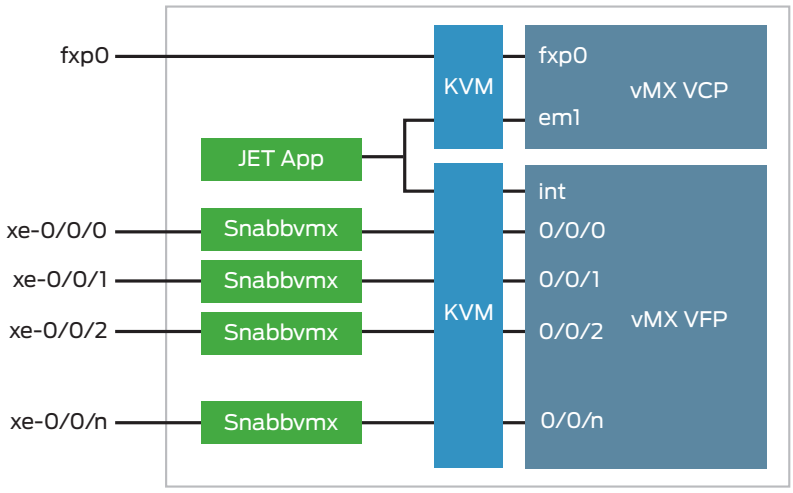

Figure 2: vMX lwAFTR Docker Container

The compute node hosts the OS with minimal software version dependencies:

- Intel 82599-based 10GbE Ethernet card
- Linux kernel version 3.10 or newer with the KVM kernel module loaded (required to run the vMX lwAFTR Docker Container on the compute node)
- Docker engine
- Hugepages

Because the container already contains Qemu, KVM, and an Ethernet card driver, there is no requirement for such elements on the compute node, nor is there a conflict if the same or different versions are present.

vMX Virtual Control Plane

The vMX VCP runs Juniper Networks Junos® operating system with the Lightweight 4over6 configuration and the operation extension loaded based on an augmented version of the IETF YANG data model[2]. Lightweight 4over6 AFTR functionality, as well as all vMX Junos OS features, are configurable via CLI and NETCONF. The Junos OS serial console is directly accessible by attaching itself to a running Docker Container ("docker attach <name>").

The management port (fxp0) of the vMX VCP is connected at runtime by Docker to the docker0 Linux bridge. The internal (em1) port of the vMX VCP is connected to an internal bridge within the running container and isolated from other containers and the compute host via network namespace isolation. This allows multiple vMX containers to run on the same compute node simultaneously.

vMX Virtual Forwarding Plane

The vMX VFP runs the virtual Trio forwarding plane software and handles packet forwarding and next-hop resolution. 4over6 tunnel termination, however, is delegated to Snabbvmx.

The internal (int) interface of the VFP is connected to the same internal bridge within the running container as em1 from VCP.

The network interfaces 0/0/0 .. 0/0/n are connected via virtio_net para-virtualized network driver in Qemu in VhostUser mode, supporting fast zero copy packet handover between the physical interface card and the guest virtual machine—in this case, vMX VFP. This requires a sufficient amount of Hugepages on the compute node.

Snabbvmx

Snabbvmx is part of the open-source virtualized Ethernet networking stack Snabb[3] and has two main functions:

1. Directly drive the physical interface port and transparently drive the transport packets to the vMX VFP
2. Perform Lightweight 4over6 AFTR tunnel termination

This dual functionality is best explained by comparing Snabbvmx to a "bump in the wire." The IPv4-in-IPv6 encapsulated traffic is decapsulated; then all nonlocal IPv4 traffic is encapsulated in accordance with the provisioned binding table. All other traffic, including ARP, IPv6 NDP, BFD, Link Layer Discovery Protocol (LLDP), OSPF, BGP, Internet Control Message Protocol (ICMP) Ping, etc., are transparently passed between the physical port and the vMX VFP virtual data interfaces (0/0/n).

The physical interface ports are referenced by their Peripheral Component Interconnect (PCI) addresses (e.g., 0000:05:00.0) at runtime. The order of the addresses given to the container at startup defines mapping to the virtual interfaces 0/0/0 through 0/0/n on the vMX. Each physical port is consumed by a Snabbvmx daemon and connected over a high-performance VhostUser[4] interface to the vMX VFP via Qemu or KVM.

If a network interface does not have a Lightweight 4over6 configuration in Junos OS, all traffic is passed transparently between the physical port and virtual data interface on the vMX virtual router. Snabbvmx is fully transparent in this case, allowing all supported vMX interface configurations, including single- and double-VLAN tagged and MPLS.

Snabbvmx has been contributed back to the Snabb project[5].

Juniper Extension Toolkit Application

The Juniper Extension Toolkit (JET) application controls the functionality of all Snabbvmx interface driver applications within the container based on the augmented Junos OS configuration for Lightweight 4over6. It also provides operational and statistical access to the Junos OS control plane. Additional information about JET can be found here.

Configuration

The interface and Lightweight 4over6 tunnel configuration is conducted via the Junos OS configuration. The mapping of physical interface ports to Junos OS interfaces is done via the ordered list of PCI interface port addresses given to the container at startup. Below is a startup example, with the available 10GbE interface ports on the compute node:

```
$ lspci |grep 10-
81:00.0 Ethernet controller: Intel Corporation 82599ES 10-Gigabit SFI/SFP+ Network
Connection (rev 01)
81:00.1 Ethernet controller: Intel Corporation 82599ES 10-Gigabit SFI/SFP+ Network
Connection (rev 01)
```

[2] https://tools.ietf.org/html/draft-ietf-softwire-yang-00
[3] https://github.com/snabbco/snabb
[4] http://www.virtualopensystems.com/en/solutions/guides/snabbswitch-qemu/
[5] https://github.com/Igalia/snabb/tree/lwaftr/src/program/snabbvmx

```
83:00.0 Ethernet controller: Intel Corporation 82599ES 10-Gigabit SFI/SFP+ Network
Connection (rev 01)
83:00.1 Ethernet controller: Intel Corporation 82599ES 10-Gigabit SFI/SFP+ Network
Connection (rev 01)

$ cat run1.sh
#!/bin/bash
NAME="lwaftr1"
CFG="lwaftr1.txt"
VMX="vmx-bundle-16.1R1.7.tgz"
CONTAINER="vmxlwaftr:v0.10"
IDENTITY="lab,lab123"
INTERFACES="0000:81:00.0 0000:81:00.1 0000:83:00.0 0000:83:00.1"
LICENSE="license-eval.txt"
docker run --name $NAME -ti --privileged -v $PWD:/u:ro $CONTAINER -i $IDENTITY -l
$LICENSE -c $CFG $VMX $INTERFACES
```

This results in the following interface mapping:

Table 2: Port to Interface and ID Mapping

PCI Address	Junos OS IFD	Snabbvmx id	lwaftr-instance id
0000:81:00.0	xe-0/0/0	xe0	0
0000:81:00.1	xe-0/0/1	xe1	1
0000:83:00.0	xe-0/0/2	xe2	2
0000:83:00.1	xe-0/0/3	xe3	3

By default, vMX uses ge-0/0/x interface naming. This can be changed to xe with the following configuration:

```
chassis {
    fpc 0 {
        pic 0 {
            interface-type xe;
        }
    }
}
```

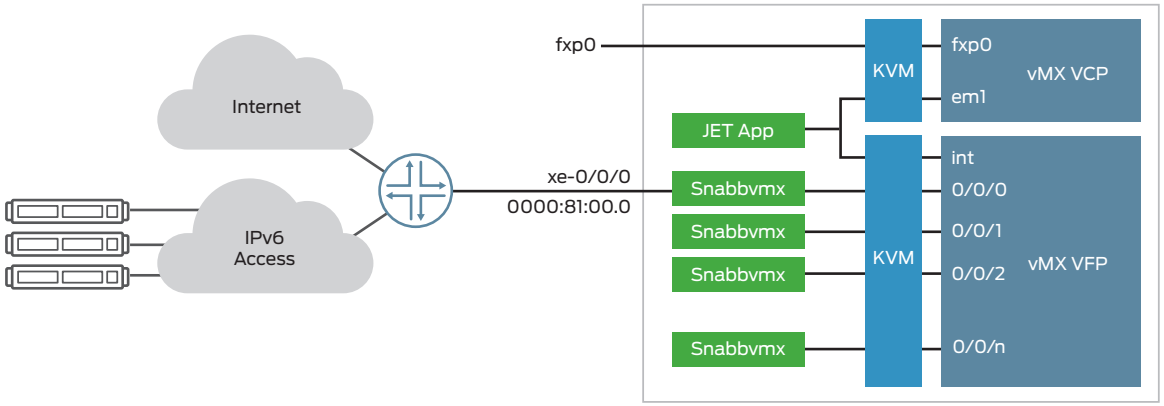

Figure 3: Example configuration with one dual-stack interface

A single physical port is connected to a router with a dual-stack configuration for IPv4 and IPv6. The vMX acts as the next hop, responding to ARP and IPv6 NDP protocol queries and IGP and BGP termination points. Optionally, BFD can also be enabled as well; however, this requires the following standard Junos OS configuration on 0/0/0:

```
interfaces {
    xe-0/0/0 {
        mtu 9000;
        unit 0 {
            description “xe0”;
            family inet {
                address 172.20.0.16/24;

            }
            family inet6 {
                address 2004:400::1/64;
            }
        }
    }
}
```

The IETF YANG schema is used to configure the Snabbvmx physical port driver with VLAN information, IPv4 and IPv6 information, as well as the Lightweight AFTR configuration and binding table. Each network interface has a corresponding lwaftr-instance object. The example configuration shows five binding entries:

```
ietf-softwire:softwire-config {
    binding {
        br {
            br-instances {
                br-instance 0 {
                    tunnel-payload-mtu 9000;
                    tunnel-path-mru 9100;
                    binding-table {
                        binding-entry 2004:400:400::1 {
                            binding-ipv4-addr 4.4.4.1;
                            port-set {
                                psid-offset 0;
                                psid 1;
                                psid-len 6;
                            }
                            brr-ipv6-addr 2004:400::1;
                        }
                        binding-entry 2004:400:400::2 {
                            binding-ipv4-addr 4.4.4.1;
                            port-set {
                                psid-offset 0;
                                psid 2;
                                psid-len 6;
                            }
                            br-ipv6-addr 2004:400::1;
                        }
```

```
                        binding-entry 2004:400:400::3 {
                            binding-ipv4-addr 4.4.4.1;
                            port-set {
                                psid-offset 0;
                                psid 3;
                                psid-len 6;
                            }
                            br-ipv6-addr 2004:400::1;
                        }
                        binding-entry 2004:400:400::4 {
                            binding-ipv4-addr 4.4.4.1;
                            port-set {
                                psid-offset 0;
                                psid 4;
                                psid-len 6;
                            }
                            br-ipv6-addr 2004:400::1;
                        }
                        binding-entry 2004:400:400::5 {
                            binding-ipv4-addr 4.4.4.1;
                            port-set {
                                psid-offset 0;
                                psid 5;
                                psid-len 6;
                            }
                            br-ipv6-addr 2004:400::1;
                        }
                    }
                    jnx-aug-softwire:ipv4_address 172.20.0.16;
                    jnx-aug-softwire:cache_refresh_interval 1;
                }
            }
        }
    }
}
```

The IPv4 address in the augmented lwaftr-instance object is used to detect local traffic that must bypass tunnel encapsulation and be passed on to the vMX network interface—in this example, xe-0/0/0.

The cache_refresh_interval, measured in seconds (if set), will periodically send a copy of the processed packet to the vMX for next-hop resolution, cache the next hop (one per family IPv4 and IPv6 each), and use it for all packets. If cache_refresh_interval is not set, all packets are sent to the vMX on default xe-0/0/0 for routing and will be forwarded to the physical port once received back.

Redundancy and Scale-Out

The stateless nature of Lightweight 4over6 at the service provider's edge allows the use of load-balancing traffic to many interfaces without the need for an encapsulation and decapsulation path going through the same interface and instance. The only requirement is that each instance have the full binding table loaded.

The vMX announces the tunnel endpoints over each interface via BGP, but with a different next hop (different IP per interface). The peering router receives four next hops per protocol family if all four instances are operational and can load-balance traffic to the interfaces using equal-cost multipath (ECMP). If one instance terminates, the route is no longer announced and the next hop is withdrawn.

vMX Interface Scale-Up

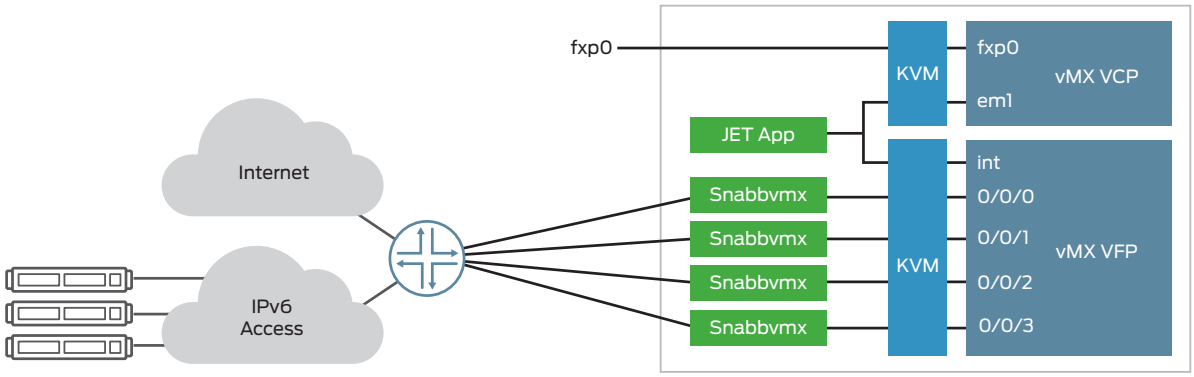

Figure 4: vMX interface scale-up

Four interfaces on a single vMX are not the actual limit. The true limit is only reached when all cores on the same non-uniform memory access NUMA nodes are in use. Traffic never traverses one interface to another, because each interface has the full binding table and hence can form encapsulation and decapsulation tunnels bidirectionally.

Multiple vMX Instances Scale-Out

The concept of interface scale-up is not limited to a single vMX. It works equally well up to the ECMP limit of the peering router, which is typically 64.

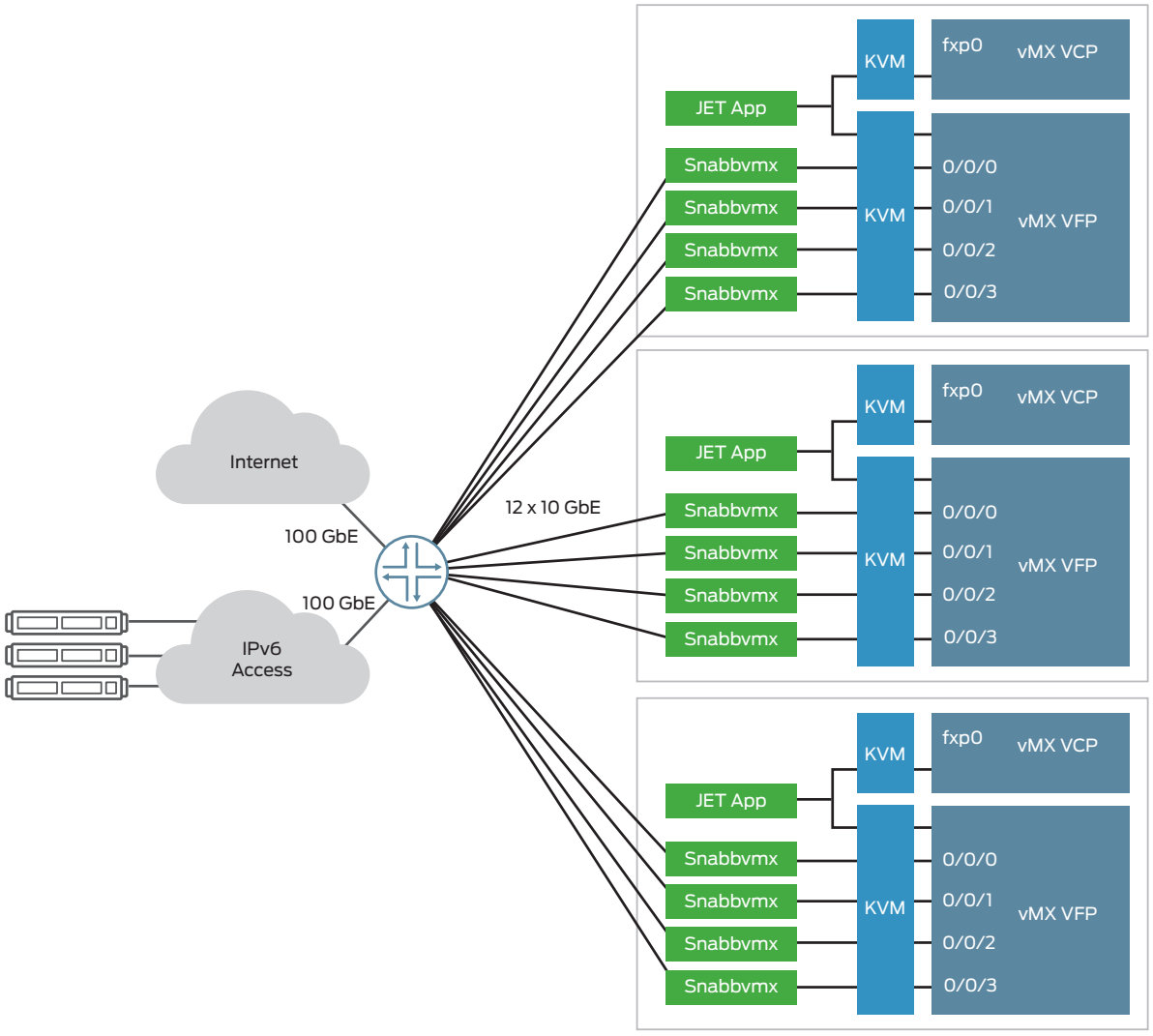

Figure 5: Multiple vMX instances of scale-out

Figure 5 shows an example of three vMX docker containers, each serving 4x10GbE, resulting in a total bidirectional capacity of 120 Gbps for millions of bindings.

Optional Dual Interface Single-Stack Design

There are networks that require the vMX to use separate physical interfaces per protocol family. Protocol encapsulation and decapsulation is still done at ingress, but next-hop caching cannot be used because the packet must be sent to the vMX for routing based on the packets' IP family.

Figure 6 shows a design using two separate physical interfaces, one for IPv4 traffic and another for IPv6 traffic. Tunnel encapsulation and decapsulation always occur at ingress, never at egress; therefore, the same binding table must be shared by both lwaftr-instances. This can be achieved by simply storing the binding table in a file and referencing it for use by all interfaces:

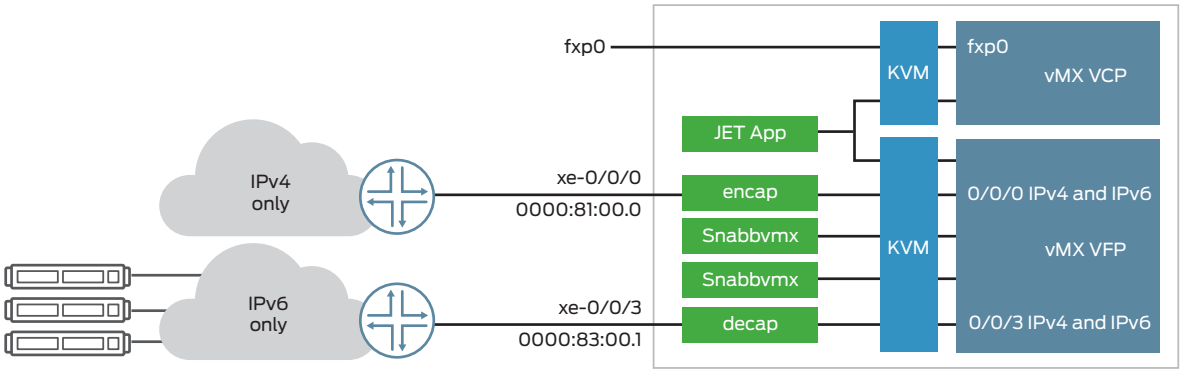

Figure 6: Dual interface single-stack design example

```
ietf-softwire:softwire-config {
    binding {
        br {
            br-instances {
                br-instance 0 {
                    tunnel-payload-mtu 9000;
                    tunnel-path-mru 9100;
                    ipv4_address 172.16.2.35;

                }
                br-instance 3 {
                    tunnel-payload-mtu 9000;
                    tunnel-path-mru 9100;
                    ipv4_address 172.16.2.35;
                }
            }
            binding-table-file binding_table_4m.txt;
        }
    }
}
```

The interfaces xe-0/0/0 and xe-0/0/1 still must be configured as dual stacks, because they have two main tasks to perform:

1. Respond to the BFD, ICMP, and peer via OSPF and BGP protocols
2. Accept and route decapsulated IPv4 packets or encapsulated IPv6 packets

IETF Softwire YANG Schema

Junos OS Release 16.1 supports a custom YANG schema to be added at runtime to extend the configuration storage, CLI commands, and access via NETCONF and gRPC.

The vMX lwAFTR Docker Container incorporates the latest draft of the IETF softwire YANG schema. Additional information can be found here.

Junos OS also supports YANG schema augmentation and deviation to add solution-specific YANG containers and leafs. For vMX lwAFTR, several additional leafs have been added and published in a separate augmentation file:

```
module jnx-softwire {
  namespace “http://yang.juniper.net/software/aug”;
  prefix “jnx-swire”;

  import ietf-inet-types {prefix inet; }
  import ietf-softwire {prefix sw; }

  organization “Juniper Networks, Inc.”;

  revision 2016-07-19 {
    description
      “Juniper software augments. “;
  }

  augment “/sw:softwire-config/sw:binding/sw:br” {
    leaf binding-table-file {
      type string;
      description “Complete path of the binding table file”;
    }
  }

  augment “/sw:softwire-config/sw:binding/sw:br/sw:br-instances/sw:br-instance” {
    leaf ipv6_address {
      type inet:ipv6-address;
    }
    leaf ipv4_address {
      type inet:ipv4-address;
    }
    leaf ipv6_vlan {
      type uint16 {
        range 0..4094;
      }
    }
    leaf ipv4_vlan {
      type uint16 {
        range 0..4094;
      }
    }
    leaf hairpinning {
      type boolean;
      default false;
    }
```

```
leaf fragmentation {
  type boolean;
  default false;
}
leaf cache_refresh_interval {
  type uint32;
  units seconds;
  default 1;
}
leaf icmpv6_rate_limiter_n_packets {
  type uint32;
}
leaf icmpv6_rate_limiter_n_seconds {
  type uint32;
  default 1;
}
leaf policy_icmpv6_incoming {
  type enumeration {
    enum allow;
    enum drop;
  }
}
leaf policy_icmpv6_outgoing {
  type enumeration {
    enum allow;
    enum drop;
  }
}
leaf policy_icmpv4_incoming {
  type enumeration {
    enum allow;
    enum drop;
  }
}
leaf policy_icmpv4_outgoing {
  type enumeration {
    enum allow;
    enum drop;
  }
}
leaf ipv6_ingress_filter {
  type string;
  description
    "IPv6 ingress filter in libpcap format";
}
leaf ipv6_egress_filter {
  type string;
```

```
      description
        "IPv6 egress filter in libpcap format";
    }
    leaf ipv4_ingress_filter {
        type string;
        description
          "IPv4 ingress filter in libpcap format";
    }
    leaf ipv4_egress_filter {
      type string;
      description
        "IPv4 egress filter in libpcap format";
    }
  }
}
```

In the event that some of these leafs are added to the final IETF RFC, they can simply be removed from the augmentation file and a new Docker Container built.

The draft-ietf-softwire YANG models have containers for features not supported by the vMX-based Lightweight 4over6 implementation, mainly ce (client side) and algorithm (for MAP-T, MAP-E). These are marked as "not-supported" by a deviation file:

```
module jnx-softwire-dev {
  namespace "http://yang.juniper.net/software/aug";
  prefix "jnx-swire";

  import ietf-softwire {prefix sw; }

  organization "Juniper Networks, Inc.";

  revision 2016-07-19 {
    description
      "Juniper software deviation (lwaftr) BR only functionality ";
  }

  deviation /sw:softwire-config/sw:binding/sw:ce {
    deviate not-supported;
  }

  deviation /sw:softwire-config/sw:algorithm {
    deviate not-supported;
  }
}
```

There are no Junos OS release dependencies other than requiring a minimal version supporting custom YANG, which is 16.1.

How Snabbvmx Works

Figure 7 shows a block diagram of all required Snabb components. IPv4 and IPv6 packets come in from the physical 10GbE port and are either passed transparently to a corresponding vMX virtual interface (e.g., xe-0/0/0) or detected as a packet in need of Lightweight 4over6 encapsulation or decapsulation. Pending configuration, fragmentation, and re-assembly can take place at ingress and egress. Snabbvmx acts as a network interface driver daemon that not only transparently passes packets between a dedicated physical 10GbE interface and a virtual interface on the vMX (running on Qemu), but also handles IPv4 in IPv6 packets and nonlocal IPv4 packets by passing them through the LwAFTR application. While the diagram only shows one Snabbvmx instance and one pair of physical and virtual interfaces, the Snabbvmx solution scales out by running multiple instances, each connecting to a separate pair of physical and virtual interfaces.

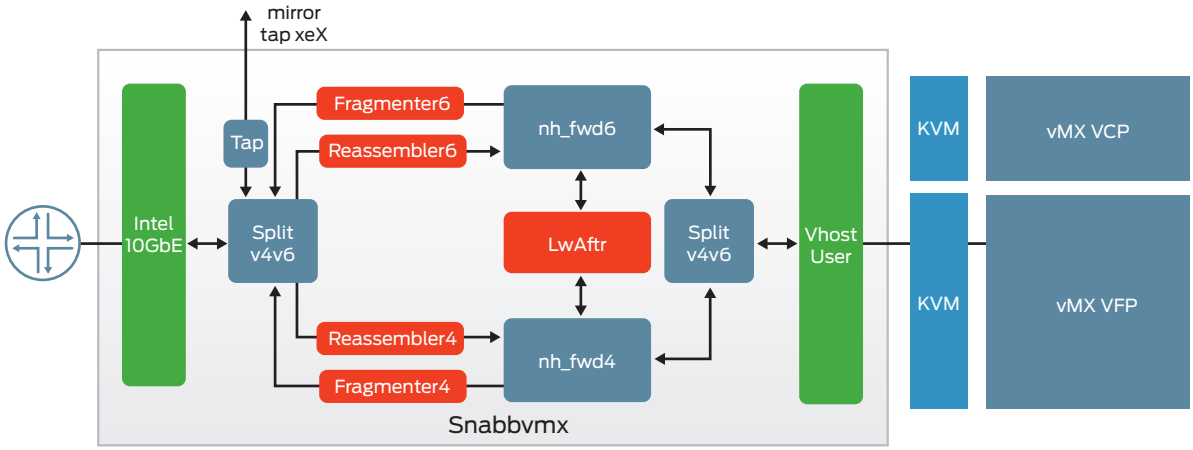

Figure 7: How Snabbvmx works internally

The Snabb daemon Snabbvmx loads the Intel10g application to drive a physical Intel 82599-based 10GbE port. The LwAFTR application is the workhorse, handling the tunnel encapsulation and decapsulation. FragmenterX and ReassemblerX are in the data path to handle IPv4 to IPv6 fragmentation and reassembly. The Split v4v6 application splits IPv4 from IPv6 packets and sends them upstream from the Intel10g application to the protocol-specific reassembler application. Packets received from the vMX via the VhostUser application are also split based on IPv4 or IPv6 and sent to the protocol-specific nh_fwd4/nh_fwd6 triage function. This nh_fwd makes a forwarding decision based on unicast or broadcast destination media access control (MAC), matching local IPv4 destination or IPv4-in-IPv6.

Split v4v6, in combination with the application tap, allows monitoring of packets coming in and out of the physical port based on matching IPv4 address, either as source, destination, or within an IPv4-in-IPv6 packet. Configuration of such filters is done via shared memory location in the file system and accessible via Snabb snabbvmx monitor <ipv4-address>.

Contrary to the operations support system (OSS) Snabb daemon for LwAFTR, Snabbvmx, ARP, IPv6 neighbor discovery, and any other control protocol like BFD or IS-IS, the vMX handles BGP and transparently passes it through Snabbvmx. This simplifies and empowers the solution for Lightweight 4over6.

The Snabbvmx daemon is controlled via configuration files that are generated by the vMX VCP. It updates binding table entries by alerting the running Snabbvmx daemon to the presence of a new, precompiled binding table. Multiple Snabbvmx daemons connected to the same vMX share the same binding table.

The configuration file for LwAFTR, fragmentation, reassembly, and binding table used internally are documented here. They are automatically generated every time the vMX Junos OS configuration is updated.

Conclusion

Lightweight 4over6 AFTR VNF based on vMX and Snabb offers a scalable, high-performance, and carrier-grade tunnel termination functionality that is easy to deploy. It scales to millions of bindings and dozens of 10GbE interfaces, limited only by the number of concurrent ECMP hops a data center gateway router supports for IPv4 and IPv6.

The Juniper Networks vMX virtual routing solution provides carrier-grade control plane and management functionality, including BGP for IPV4 and IPv6, BFD, and custom YANG for provisioning according to published IETF softwire schema. Meanwhile, Snabb provides high-performance Lightweight 4over6 tunnel encapsulation and decapsulation with optional fragmentation and re-assembly for millions of bindings. The combination of vMX and Snabb gives service providers the flexibility to migrate to IPv6 networks while ensuring service continuity for IPv4 users.

About Snabb

Snabb (formerly "Snabb Switch") is a simple and fast packet networking toolkit, publically available under the Apache 2.0 license at https://github.com/snabbco/snabb.

Snabb is written using these main techniques:

- Lua, a high-level programming language that is easy to learn
- LuaJIT, a just-in-time compiler that is competitive with C
- Ethernet I/O with no kernel overhead ("kernel bypass" mode)

Snabb compiles into a standalone executable called snabb. This single binary includes multiple applications and runs on any modern Linux/x86-64 distribution.

Snabb lwAFTR implements the Internet-facing AFTR component of Lightweight 4-over-6 (lw4o6) and can also be run standalone, though without NETCONF/YANG and without dynamic routing control plane. For more information, visit https://github.com/snabbco/snabb/tree/master/src/program/lwaftr/doc.

Snabbvmx is currently being contributed back into the Snabb project. See https://github.com/Igalia/snabb/pull/391 for current status.

About Juniper Networks

Juniper Networks challenges the status quo with products, solutions and services that transform the economics of networking. Our team co-innovates with customers and partners to deliver automated, scalable and secure networks with agility, performance and value. Additional information can be found at Juniper Networks or connect with Juniper on Twitter and Facebook.

Corporate and Sales Headquarters
Juniper Networks, Inc.
1133 Innovation Way
Sunnyvale, CA 94089 USA
Phone: 888.JUNIPER (888.586.4737)
or +1.408.745.2000
Fax: +1.408.745.2100
www.juniper.net

APAC and EMEA Headquarters
Juniper Networks International B.V.
Boeing Avenue 240
1119 PZ Schiphol-Rijk
Amsterdam, The Netherlands
Phone: +31.0.207.125.700
Fax: +31.0.207.125.701

2000648-001-EN Nov 2016