Evaluation of virtualization and traffic filtering methods for container networks

Łukasz Makowski[a], Cees de Laat[a], Paola Grosso[a,*]

[a]*System and Network Engineering Group (SNE), University of Amsterdam, Science Park 904, Amsterdam, The Netherlands*

Abstract

Future distributed scientific applications will rely on containerisation for data handling and processing. The question is whether container networking, and the associated technologies, are already mature enough to support the level of usability required in these environments. With the work we present in this article we set out to experiment and evaluate three novel technologies that support addressing and filtering: EVPN, ILA and Cilium/eBPF. Our evaluation shows that different level of maturity, with EVPN more suitable for adoption. Our work also indicates that to support true multi-tenancy further integration of addressing technologies and filtering technologies is needed.

Keywords: containers, overlay networks, packet filtering, addressing, multi-tenancy

1. Introduction

Many business applications use containers to perform complex or distributed computational tasks. The use of containers for scientific applications is also becoming more and more evident.

This increased adoption goes hand in hand with new technical challenges for the operations of distributed containers systems. Some of these new challenges are intrinsic to the nature of scientific cooperation, which relies on often on communication and cooperation between different scientists in different institutions. An area that has started to get significant attention is container networking[1][2]; a number of shortcomings have been identified, and they need to be properly addressed before containers can be heavily deployed by scientists in distributed settings.

A first challenge relates to addressing. Assuming a container uses the hosts' IP address for outside communication, it is feasible to forward needed requests based on a well-known application port. Nevertheless, this approach becomes problematic when multiple containers use the same port number, as it implies the use of techniques such as port-level NAT or proxying to forward a request to its destination. A way to remove this overhead would be assigning an IP address to every container in a system, however, this introduces new problems e.g. IP address mobility or exhaustion of addressing space. Another option is creating a virtual network atop of physical network, commonly known as an overlay. In this research we looked at two new technologies: EVPN (Ethernet VPN) and ILA (Identifier Locator Addressing) for providing virtual networks. Our focus has been on assessing their respective capability to support a flexible environment for container systems. First, we analysed EVPN and ILA in a side-by-side fashion, in regard to the data and control plane capabilities supporting multi-tenancy. Next, using available projects we built rudimentary proof-of-concept environments utilising EVPN and ILA for container communication.

A second challenge relates to load balancing and filtering traffic. EVPN and ILA are not suitable for this more complex tasks. A new project, Cilium (`https://www.cilium.io/`), is addressing these problems, and we looked at its performance in regard to filtering compared to a Docker Swarm (`https://docs.docker.com/engine/swarm/`) based container overlay.

In the following section we will briefly cover the general features of container networks, and provide a more detailed introduction to EVPN, ILA and Cilium (Sec. 2). We will first present the experiments we conducted to compare ILA and EVPN (Sec. 3); we will follow with the tests made for the performance evaluation of Cilium (Sec. 4). We will present a summary of our findings (Sec. 5); and then compare our work with the related work in the area (Sec. 6). In Sec. 7 we identify future direction of work based on the current results.

2. Background

2.1. Container networking

The concept of overlay networks is not a recent invention, but in the virtualization era it is getting more and more attention. The IETF's Network Virtualization Overlays (nvo3) working group has produced a number of documents discussing design and architecture of network virtual overlays (NVOs). Lasserre et al. [3] provided the NVO fundamental functional components, which are illustrated in Fig. 1.

*Corresponding author

Email addresses: `makowski@uva.nl` (Łukasz Makowski), `delaat@uva.nl` (Cees de Laat), `pgrosso@uva.nl` (Paola Grosso)

[1] https://tools.ietf.org/html/rfc7365

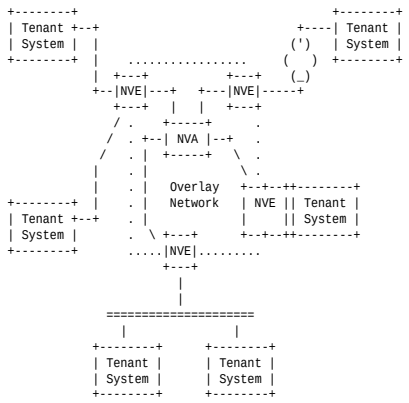

Figure 1: NVO reference model[1]

This reference model distinguishes between tree elements:

- Network Virtualization Edge (NVE). The tunnel endpoint, realizing the features such as encapsulation and providing so called Virtual Network (VN) context allowing to distinguish the traffic of different network instances.

- Network Virtualization Authority (NVA). The component providing an information about virtual endpoint reachability in the overlay. Precisely, informing an NVE how a packet should be addressed so that it will make its way through an underlay network.

- Tenant System (TS). An entity belonging to the network tenant, attached to one or more VN instances e.g. VM.

Narten et al. [4] emphasize multitenancy as a major feature for a modern NVOs. The important attribute of virtual network is the ability to isolate particular VN instances from each other, while being TS addressing agnostic.

ILA is inspired with other Identifier/Locator protocols such as ILNP [13] and LISP [14]. Nonetheless, the ILA creators argue that unlike ILNP it is easier to deploy in existing infrastructures as it does not require new encapsulation and compatibility with existing network equipment.

ILA does not use any encapsulation, instead, it uses left 64 bits of an address as a Locator (identifying physical node) and the remaining 64 bit part (Identifier) to uniquely address an endpoint.

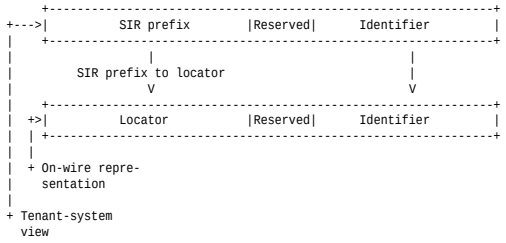

Figure 3: ILA address translation overview. A SIR prefix in packet's destination address is replaced with a Locator before it is transmitted across an NVO.

The endpoints (e.g. containers) in ILA network are unaware of the Locator part of an address. While they still use IPv6 addresses to communicate, ILA packet destination address is a subject to a translation. From endpoint's perspective the address, consists of the SIR prefix concatenated with the Identifier. The SIR remains virtual, getting replaced with the Locator every time the actual ILA packet enters the wire (Fig. 3). Hence to this process, the endpoint mobility is possible as the address is not tied to an underlay and can be placed anywhere (within ILA network).

The ILA authors also published more deployment focused document [15]. They discuss the approaches to integrate ILA with a multiple Linux kernel namespaces which is useful considering our goal of using ILA as a container network. Although, ILA standard definition does not enforce any particular solution to be used as an NVA, there is a recent draft document [16] defining required MP-BGP [17] extensions, showing that ILA could follow the approach known from EVPN.

2.4. Cilium

Cilium leverages eBPF as technology for traffic filtering and policy definition. The eBPF [18] Linux kernel mechanism has a potential to enhance a traditional way of packet filtering with the aid of netfilter [19][20]. The eBPF's predecessor — classical BPF (cBPF) [21] was created to improve on the ability to filter packets implemented by embedding an in-kernel VM interpreting machine-level filtering instructions. eBPF has further extended the available instruction set enabling the creation of more sophisticated programs. Since that time, it has been used for various purposes such as system performance analysis [22] or DDoS protection [23].

Currently, a single eBPF program has a limitation of 4096 instructions, however, as it can tail-call (maximum 32 times) another program it is not a functionality blocking restriction. Because eBPF code is stateless by design, a special construct (maps) has been created to allow saving desired information to be accessed by further program invocations. eBPF maps work in key/value fashion and can also be accessed from user-space. A single program can use 64 maps at maximum. In Fig. 4 we illustrate how an eBPF program can be used. The common way of creating eBPF programs is using languages like C or P4, which with the aid of LLVM[2] can be compiled into ELF format. Once compiled, an eBPF program can be used in-kernel by using a *bpf()* system call. Basically, after the program passes the verification it might be attached to a hook point e.g. Linux traffic control (TC) sub-system.

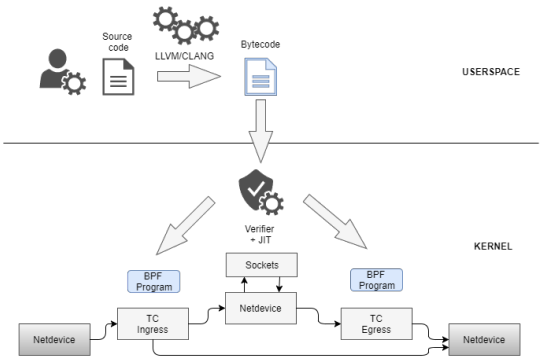

Figure 4: Overview of BPF kernel injection.

TC as a point of attachment allows performing actions on packets before entering the kernel's networking stack. The available actions are: content rewrite; size trimming/extension and redirection to other network devices.

Cilium heavily relies on eBPF technology for performing packet related operations, significantly reducing code-path network packet structure traverses in the kernel. Cilium generates eBPF programs which are further hooked to Linux traffic control (TC) sub-system, which results in an eBPF program acting in place of network interface packet queueing sub-routine (Listing 1).

```
~# tc filter show dev lxc20156 ingress
filter protocol all pref 41159 bpf
filter protocol all pref 41159 bpf handle 0x1 bpf_lxc.o:[from-container]
direct-action

~# tc filter show dev cilium_vxlan ingress
filter protocol all pref 49152 bpf
filter protocol all pref 49152 bpf handle 0x1 bpf_overlay.o:[from-overlay]
direct-action
```

Listing 1: Cilium generated eBPF programs (`bpf_lxc.o`, `bpf_overlay.o`) are used to handle the container-overlay communication path.

Cilium decouples addresses from policies, using the concept of labels and identifiers. Every container connected to Cilium is assigned with a label, which makes it part of a group of containers. Later on, an operator may decide to specify rules (policy) how those groups (identified by a label) can communicate. Such rule is being automatically assigned with an ID called an identity, which is allocated to any container affected by it (Listing 2).

[2]https://llvm.org/
[2]http://wiki.osdev.org/ELF6

```
root@cilium-master:~# cilium endpoint list
ENDPOINT IDENTITY LABELS (source:key[=value]) IPv6 IPv4 STATUS
14594 291 cilium:id.service1 fd02::c0a8:210b:0:3902 10.11.20.8 ready
20084 291 cilium:id.service1 fd02::c0a8:210b:0:4e74 10.11.58.218 ready
```

Listing 2: Listing of endpoints (containers) connected to Cilium overlay. Both endpoints (`14594`, `20084`) are assigned with the same label (`cilium:id.service1`) and share the same identity (`291`).

In its VXLAN flavor, Cilium uses VNID field to a network packet with identity it belongs to. This enables the receiving container host to ignore incoming packets' characteristics (protocol, port or address) and conduct matching only on a

3. ILA and EVPN comparison

The first step of our work has been a comparison of the technologies based on the documentation available (see Sec. 3.1). A second step has been the creation of two distinct test environments (see Sec. 3.2 and Sec. 3.3) to assess the ease of use of the two technologies, which we describe in Sec. 3.4

3.1. Multitenancy comparison

The result of this analysis are shown in Fig. 5.

Feature	EVPN	ILA
type	L2/L3	L3
data-plane	VXLAN, MPLS	IPv6 address Locator/Identifier
control-plane	MP-BGP	not defined
VN context	VNID, RT/RD[3]	SIR[4], VNID

Figure 5: EVPN-ILA feature comparison

To be able to create multiple virtual networks the packet format should provide information (in the form of VNID) which context a given packet belongs to. The size of the VNID space which is considered to be feasible to meet the majority of demands is one million values (20 bits) [24]. Both, EVPN with VXLAN and ILA fulfill this requirement. In the VXLAN case this is quite straightforward as it uses its dedicated packet field (20 bits) for that purpose, however, ILA (given its specifics) is less intuitive. It dedicates 28 bits of Identifier part to be used as a VNID, which can be further used in the process of translation between ILA's tenant virtual prefix and on-the-wire representation (see Fig. 6).

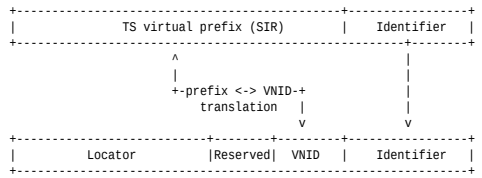

Figure 6: ILA multitenancy concept. For every packet a VNID can be translated to tenant virtual prefix.

Such approach enables the possibility to host tenants and providing the means to isolate the traffic in a scope of a single VNID. However, it does not provide the flexibility in regard to tenant system virtual prefix selection. As the VNID field is used to map traffic to a particular client, colliding TS virtual prefix would result in ambiguity. For example, if VNID1 points to SIR1, another tenant's VNID2 to SIR1 mapping is conflicting.

In contrast, EVPN is capable of providing both the isolation, as well as freedom in tenant's address selection. As the VXLAN VNID tag does not interfere with a packet it encapsulates, there is no issue with having overlapping addresses as long as those are put within distinct VXLAN segments.

3.2. EVPN test environment

To prototype a EVPN network we used the components and the general idea described in [25].

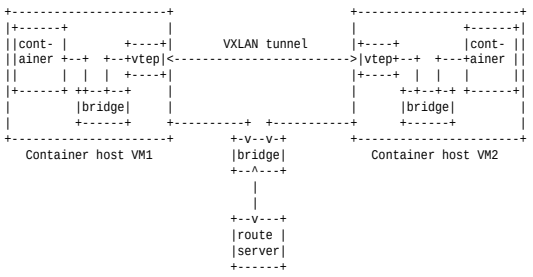

Figure 7: EVPN environment topology

As shown in Fig. 7, the topology consisted of two VMs operating as a container hosts and an additional one running as a route-server. Each container host was running a BGP daemon (bagpipe-bgp[5]) peering with a route-server (GoBGP[6]). Once a container was started, bagpipe-bgp created EVPN route Type-2 (Listing 3) containing its IP and MAC addresses. Upon processing the BGP update the receiving container host was able to set-up its side of a VXLAN tunnel using "Route Distinguisher" (line 24; containing IP of a container host originating the update) and "MPLS Label Stack: 136" (line 31; in this case, representing VXLAN tunnel ID) parameters.

```
Border Gateway Protocol - UPDATE Message
    Marker: ffffffffffffffffffffffffffffffff
    Length: 107
    Type: UPDATE Message (2)
    Withdrawn Routes Length: 0
    Total Path Attribute Length: 84
    Path attributes
        Path Attribute - ORIGIN: IGP
        Path Attribute - AS_PATH: empty
        Path Attribute - LOCAL_PREF: 100
        Path Attribute - EXTENDED_COMMUNITIES
        Path Attribute - MP_REACH_NLRI
            Flags: 0x80, Optional: Optional, Non-transitive, Complete
            Type Code: MP_REACH_NLRI (14)
            Length: 48
            Address family identifier (AFI): Layer-2 VPN (25)
            Subsequent address family identifier (SAFI): EVPN (70)
            Next hop network address (4 bytes)
            Number of Subnetwork points of attachment (SNPA): 0
            Network layer reachability information (39 bytes)
                EVPN NLRI: MAC Advertisement Route
                    AFI: MAC Advertisement Route (2)
                    Length: 37
                    Route Distinguisher: 0001c0a8320b0094 (192.168.50.11:148)
                    ESI: 00 00 00 00 00 00 00 00 00 00
                    Ethernet Tag ID: 0
                    MAC Address Length: 48
                    MAC Address: 52:da:46:13:7e:7d (52:da:46:13:7e:7d)
                    IP Address Length: 32
```

[5]https://github.com/Orange-OpenSource/bagpipe-bgp
[6]https://osrg.github.io/gobgp/

```
30                  IPv4 address: 10.1.2.8
                    MPLS Label Stack: 136 (bottom)
```

Listing 3: EVPN route Type-2 update indicating presence of a container with IP 10.1.2.8 (line 30) and MAC 52:da:46:13:7e:7d (line 28).

3.3. ILA test environment

The ILA environment (Fig. 8) consisted of two container host VMs interconnected with a bridge. To provide the ability to create ILA tunnels we configured three types of IPv6 addresses (Listing 4):

- address configured directly on container host interface used for external communication (providing regular IPv6 reachability)
- ILA locator prefix (e.g. /64), which needed be reachable across the whole ILA environment
- ILA SIR address residing within a container namespace (used for container to container communication)

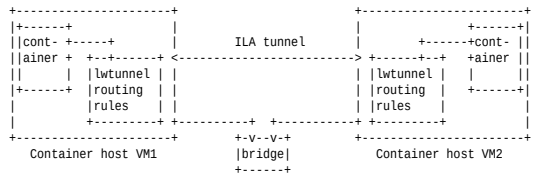

Figure 8: Test topology

```
root@ila-1:/vagrant/# ip -6 a l
#(output omitted for brevity)
3: enp0s8: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 state UP qlen 1000
  inet6 2803:6082:1950:401:2555:0:1:0/64 scope global deprecated
# +---Locator-addr.--------------+
  valid_lft forever preferred_lft 0sec
  inet6 2001:610:158:2600::1/64 scope global
# +---Regular-addr.------+
  valid_lft forever preferred_lft forever
  inet6 fe80::a00:27ff:fe45:5ebb/64 scope link
  valid_lft forever preferred_lft forever
5: veth0@if4: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 state UP qlen 1000
    inet6 face:b00c::2555:0:0:1/64 scope global
#         +-----SIR-addr.-----+
       valid_lft forever preferred_lft forever
```

Listing 4: ILA addressing setup

To provide the actual ILA functionality we used Linux kernel's "ila" module (source version 7FDBACFBFD562604DFB0735). Although ILA does not use encapsulation, conceptually it also utilizes tunneling approach for the communication. Authors of Linux kernel module implemented this with the aid of lightweight tunnel (lwtunnel) feature. Lwtunnel does not create a special tunnel device as it happens in case of VXLAN, instead, the tunnel can be used by creating an IP route and supplying the "encap ila"" keyword. As a regular IP route is matched against packet's destination, for the actual bi-directional ILA tunnel operations needed to be two routes present: one for egress and second for ingress packet flow (Listing 5).

```
#egress route
root@ila-1:/vagrant# ip -6 route show  | grep ila
face:b00c::2555:0:2:0 encap ila 2803:6080:8960:4473  csum-mode no-action \
#+-Remote SIR addr.-+          +--Remote-locator-+
via 2001:610:158:2600::2 dev enp0s8 metric 1024  pref medium
#+--Gateway's-IP-------+
#ingress route
root@ila-1:/vagrant# ip -6 route show table local | grep ila
2803:6082:1950:401:2555:0:1:0 encap ila face:b00c:0:0 csum-mode no-action \
#+-----Remote-SIR-addr.-----+          +---SIR-----+
via face:b00c::2555:0:1:0 dev veth0 metric 1024  pref medium
#   +--Dst-SIR-addr-----+
```

Listing 5: ILA tunnel routes

3.4. Ease-of-use

Our efforts to create two working environment for EVPN and ILA provided us a good insight in the current status of these two technologies.

To build an operational EVPN set-up we were able to use standard components to realize data and control planes. This allows to concentrate our focus on configuring BGP sessions, as well as on the actual components integration and not the modifications to the code.

In contrast, creating an operational ILA set-up consumed much more effort and resources than EVPN one. First, there was no actual documentation of the used kernel module. This resulted in an attempt to recreate the configuration presented by one of the authors [26]. We also had to discover by trail-and-error that there are a number of side effect of ILA such as:

- disabling packet checksumming on a container interface for UDP/TCP communication
- removing kernel created local Locator route breaking ingress ILA translation

In essence with ILA we did not reach the stage of getting this programmatically integrated, which is a pre-requisite to using a control-plane. This resulted in a need to manually create ILA tunnels between the containers.

4. Cilium filtering performance

We decided that using physical servers in the role of container hosts can potentially provide close to real world results regarding network performance. For the purpose of analyzing Cilium we built a set-up (Fig. 9) consisting of two Supermicro X8DTT-H servers equipped with Intel Xeon CPUE5620 with 24G DDR3 1066 MHz of RAM. Servers were installed with Ubuntu 17.04 operating system, Docker container engine version 17.05.0-ce (build 89658beand) and Cilium version 0.9.90 (build 08c1e0c4).

The MTU of the link interconnecting the servers was set to 9000 bytes. Furthermore, we also adjusted virtual components settings:

- MTU of virtual bridges and VXLAN/veth interfaces was set to 8950 bytes
- we enabled the Generic Segmentation Offload (GSO), Generic Receive Offload (GRO), TCP Segmentation Offload (TSO) features of veth and VXLAN devices
- on physical network interfaces, we enabled GSO and GRO, while disabling TSO

Figure 9: Netfilter/Cilium topology

We decided to compare two possible approaches one could take to enforce policies in a container overlay:

1. Docker Swarm overlay and netfilter. Using Docker Swarm 'overlay' driver for creating a virtual network and using netfilter as a packet filter.

2. Cilium overlay. Interconnecting containers with Cilium's provided overlay capability combined with usage of Cilium's traffic policies.

For each of those setups we created two scenarios. The first one was targeted to observe the baseline performance of the environments. Specifically, we measured the performance without any policy/filtering rules applied. Secondly, we conducted the experiments filtering the incoming packets based on destination TCP port. In Listing 6 we present the way we implemented this with netfilter, whereas Listing 7 illustrates the Cilium case.

```
ip netns exec 1-s8idcnjdiq iptables -t filter -A FORWARD -m state --state
      ESTABLISHED,RELATED -j ACCEPT
ip netns exec 1-s8idcnjdiq iptables -t filter -A FORWARD -m tcp -p tcp --dport
      5201 -j ACCEPT
ip netns exec 1-s8idcnjdiq iptables -t filter -P FORWARD DROP
```

Listing 6: Netfilter filtering policy applied withing the namespace of Docker overlay network (1-s8idcnjdiq).

```
[{
    "endpointSelector": {"matchLabels":{"id":"service1"}},
    "ingress": [{
        "fromEndpoints": [
            {"matchLabels":{"id":"service1"}}
        ],
        "toPorts": [{
                "ports": [{"protocol": "tcp", "port" : "5201"}]
        }]
    }]
}]
```

Listing 7: Cilium filtering policy JSON file.

In total, we performed 4 different types of measurements. The common element was that we performed 60 seconds long "iperf3" TCP bandwidth measurements (N=100) between two containers placed on distinct hosts.

4.1. Filtering comparison

In our experiments we measured the bandwidth speed of a TCP stream between two containers interconnected with an network overlay. The results of conducted performance tests are depicted in Fig. 10. There are four bars in total, each for one type of measurement performed:

- ipt_no_rules, Docker Swarm overlay without any filter in place
- cilium_no_rules, Cilium overlay without any filter in place
- ipt_tcp5201, Docker Swarm overlay with iptables/netfilter rules
- cilium_tcp5201, Cilium with its policies in place

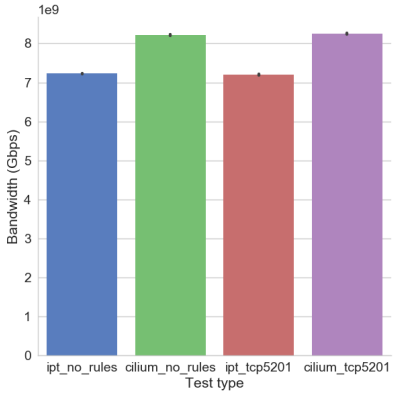

Figure 10: Iptables and Cilium based filtering comparison.

For two first cases (ipt_no_rules, cilium_no_rules) we measured the average speed of 7.22 Gbps, whereas for Cilium an equivalent test scored 8.22 Gbps on average. That shows that Cilium based network performed noticeably better than Docker Swarm created one.

In the scenario with the filtering (ipt_tcp5201, cilium_tcp5201), we observed that there was no noticeable filtering performance effect on the speed for the corresponding tests. In specific, we reached 7.20 Gbps for Docker Swarm overlay and 8.24 Gbps with Cilium.

5. Discussion

EVPN and ILA evaluation

The EVPN and ILA test environments we created were capable of performing basic operations, however, we consider them to be still in a preliminary stage. The EVPN and ILA setups lack the integration with a container orchestrator so that multiple virtual networks spanning over many container hosts can be deployed. Moreover, ILA requires also a control-plane solution, making this technology even less suitable for widespread adoption.

Cilium performance advantage

As our results showed in Sec. 4.1 the eBPF-accelerated Cilium demonstrates noticeably higher bandwidth than an overlay setup based on Docker Swarm. Interestingly, the filtering capabilities of the former are not playing the key role in those results. We believe instead that the fact Cilium implements the packet's path mainly in eBPF, as we described in Sec. 2.4, is the main source of the observed performance benefit.

Moving to the filtering impact itself, there was no noticeable effect of the policies we implemented. Arguably, the prevailing majority of the traffic we generated in our experiment was matched against the first entry in the defined netfilter rule-set.

Thanks to that we avoided the need for per-packet sequential rule evaluation, which it is in turn known to lead to the performance degradation. A further exploration on this topic should include the use of parallel streams between multiple containers, while using higher capacity physical links between the servers. Such a setup could reveal more border conditions which we did not reach with the established design.

6. Related work

Del Piccolo et al. [27] conducted a comprehensive study over the available solutions enabling mobility and multi-tenancy in a data-center. The study compares fifteen different approaches in regard to complexity, overhead, resiliency, scalability and the ability to span multiple sites. EVPN is mentioned in the context of empowering VXLAN or NVGRE tunnels creation, however, there is no discussion over its actual use for creating a data-center fabric. Moreover, due to its novelty ILA is not included at all in the analysis presented in the article. Lastly, the central point of discussion is about VM-oriented connectivity, whereas in our work we specifically look at the use of containers as customer endpoints.

Guenender et al. [28] propose a new overlay approach called *NoEncap*. The authors discuss about the disadvantages and redundant data intrinsic to the traditional encapsulation based overlays. They identify the major design principles for *NoEncap*; after that they compare the performance of their implementation with VXLAN encapsulation. The presented experiments consist of throughput and latency measurements between pairs of virtual machines residing on physical servers interconnected with a 40 Gbps link. The results show that the proposed approach outperforms the software VXLAN overlay for both of the collected metrics, oscillating relatively close to a native (non-overlay) communication performance. Based on this research, we see a solid motivation for evaluating ILA technology and enriching the landscape of encapsulation-less overlays with this new approach.

Although the performance of various netfilter scenarios and the effect of a number of rules has been studied [29] [30], there was not such evaluation performed in the context of overlay networks for containers. Although Jouet et al. [31] used BPF to enhance on the packet matching in OpenFlow, we thought more instructive and worthwhile to look at a side-by-side comparison of netfilter and eBPF when they are applied to an identical workload.

7. Conclusions and future work

The adoption of containers for distributed and multi-domain scientific applications will require the creation of suitable container networks. This will require to solve problems related to addressing and filtering that emerge in these multi-tenants environment.

Our work, presented in this article, initiates a through evaluation of a number of emerging technologies that can help to solve these problems. We have focused on EVPN and ILA for addressing, and on Cilium/eBPF for filtering. Our experimentations have allowed us to draw some general conclusion that we think are interesting for the networking community looking to support containers networks for science.

First, we believe that EVPN can be adopted to be used as a container network. Data and control plane options defined by the standard are commonly available, which significantly shortens time required to bootstrap. Whereas, ILA has only data-plane implementations which make it practically unusable as a virtual network at the current stage. Nevertheless, both solutions we presented will require extra work to integrate them with a container orchestration platform of choice.

After our analysis we find EVPN to have more potential regarding multi-tenancy than ILA. The former can host endpoints with conflicting addressing spaces without any constraints, whereas ILA requires unique SIR and Identifier components across all tenants.

Second, our Cilium network evaluation shows that eBPF technology can deliver a better performance while still maintaining the functionality of a regular VXLAN based overlay. In our performance evaluation, we did not observe notable effect of filtering performance for either Cilium or netfilter.

In the coming months we will continue to work on extending our EVPN and ILA setups with additional focus on multi-tenancy in such virtual networks. Furthermore, we will draw from the lessons learned from our Cilium analysis in order to integrate the above with performance-effective traffic filtering.

Lastly, we believe that container networks will be effective for supporting science if they will exhibit Service Function Chaining (SFC)-like behaviour. We will work on expanding current container networking approaches with solutions allowing the above. Specifically, we aim at easing the interaction between different VN instances in such a way that load-balancing and filtering could be achieved. We see the route advertisement methods that has been recently been proposed in the IETF [32] as a promising and scalable way of approaching this problem.

8. Acknowledgments

This work is funded by a grant from SURFnet in the program Research on Networks (2017). Special thanks go to Ronald van der Pol and Marijke Kaat for their support during this work. We are also indebted to Tako Mars and Nick de Bruijn that paved the way for this reseearch during their master graduation projects.

[1] J. Claassen, R. Koning, P. Grosso, Linux containers networking: Performance and scalability of kernel modules, in: Network Operations and Management Symposium (NOMS), 2016 IEEE/IFIP, IEEE, 2016, pp. 713–717.

[2] T. Yu, S. A. Noghabi, S. Raindel, H. H. Liu, J. Padhye, V. Sekar, Freeflow: High performance container networking., in: HotNets, 2016, pp. 43–49.

[3] M. Lasserre, F. Balus, T. Morin, N. Bitar, Y. Rekhter, Framework for Data Center (DC) Network Virtualization, rFC7365 (October 2014).
URL `http://tools.ietf.org/rfc/rfc7365.txt`

[4] T. Narten, E. Gray, D. Black, L. Fang, L. Kreeger, M. Napierala, Problem Statement: Overlays for Network Virtualization, rFC7364 (October 2014).
URL `http://tools.ietf.org/rfc/rfc7364.txt`

[5] K. Kompella, Virtual Private LAN Service (VPLS) Using BGP for Auto-Discovery and Signaling, rFC4761 (January 2007).
URL https://tools.ietf.org/html/rfc4761

[6] E. Rosen, BGP/MPLS IP Virtual Private Networks (VPNs), rFC4364 (February 2006).
URL https://tools.ietf.org/html/rfc4364

[7] A. Sajassi, BGP MPLS-Based Ethernet VPN, rFC7432 (February 2015).
URL https://tools.ietf.org/html/rfc7432

[8] M. Mahalingham, Virtual eXtensible Local Area Network (VXLAN): A framework for overlaying virtualized layer 2 networks over layer 3 networks, rFC7348 (August 2014).
URL https://tools.ietf.org/html/rfc7348

[9] X. Xu, N. Sheth, L. Yong, R. Callon, D. Black, Encapsulating MPLS in UDP, rFC7510 (April 2015).
URL http://tools.ietf.org/rfc/rfc7510.txt

[10] T. Worster, Y. Rekhter, E. Rosen, Encapsulating MPLS in IP or Generic Routing Encapsulation (GRE), rFC4023 (March 2005).
URL http://tools.ietf.org/rfc/rfc4023.txt

[11] A. Sajassi, A Network Virtualization Overlay Solution using EVPN, draft-ietf-bess-evpn-overlay-08 (March 2017).
URL https://tools.ietf.org/html/draft-ietf-bess-evpn-overlay-08

[12] T. Herbert, Identifier-locator addressing for IPv6, draft-herbert-nvo3-ila-04 (March 2017).
URL https://tools.ietf.org/html/draft-herbert-nvo3-ila-04

[13] R. Atkinson, Identifier-Locator Network Protocol (ILNP) Architectural Description, rFC6740 (November 2012).
URL https://tools.ietf.org/html/rfc6740

[14] D. Farinacci, The Locator/ID Separation Protocol (LISP), rFC6742 (Januari 2013).
URL https://tools.ietf.org/html/rfc6742

[15] P. Lapukhov, Deploying Identifier-Locator Addressing (ILA) in datacenter networks, Internet-Draft draft-lapukhov-ila-deployment-01, Internet Engineering Task Force, work in Progress (Oct. 2016).
URL https://datatracker.ietf.org/doc/html/draft-lapukhov-ila-deployment-01

[16] P. Lapukhov, Use of BGP for dissemination of ILA mapping information, draft-lapukhov-bgp-ila-afi-02 (October 2016).
URL https://tools.ietf.org/html/draft-lapukhov-bgp-ila-afi-02

[17] T. Bates, Multiprotocol Extensions for BGP-4, rFC4760 (January 2007).
URL https://tools.ietf.org/html/rfc4760

[18] A. Starovoitov, The Berkeley packet filter, https://www.kernel.org/doc/Documentation/networking/filter.txt.

[19] P. Ayuso, Netfilter's connection tracking system, LOGIN: The USENIX magazine 31 (3).

[20] P. Russel, H. Welte, Netfilter Hacking How-to.
URL https://www.netfilter.org/documentation/HOWTO/netfilter-hacking-HOWTO.txt

[21] S. McCanne, V. Jacobson, The BSD Packet Filter: A New Architecture for User-level Packet Capture, http://www.tcpdump.org/papers/bpf-usenix93.pdf, visited on: 07-06-2017 (December 1992).

[22] IOVisor community, BPF-based Linux IO analysis, networking, monitoring, https://github.com/iovisor/bcc.

[23] G. Bertin, XDP in practice: integrating XDP into our DDoS mitigation pipeline.

[24] D. Black, J. Hudson, L. Kreeger, M. Lasserre, T. Narten, An Architecture for Data-Center Network Virtualization over Layer 3 (NVO3), rFC8014 (December 2016).
URL http://tools.ietf.org/rfc/rfc8014.txt

[25] M. Mukhtarov, Experiments with container networking: Part 2 (2016).
URL http://murat1985.github.io/kubernetes/cni/2016/05/15/bagpipe-gobgp.html

[26] P. Lapukhov, Internet-scale Virtual Networking using ILA, https://www.nanog.org/sites/default/files/20161018_Lapukhov_Internet-Scale_Virtual_Networking_v1.pdf, retrieved on: 05-06-2017 (October 2016).

[27] V. D. Piccolo, A. Amamou, K. Haddadou, G. Pujolle, A survey of network isolation solutions for multi-tenant data centers, IEEE Communications Surveys Tutorials 18 (4) (2016) 2787–2821. doi:10.1109/COMST.2016.2556979.

[28] S. Guenender, K. Barabash, Y. Ben-Itzhak, A. Levin, E. Raichstein, L. Schour, Noencap: Overlay network virtualization with no encapsulation overheads, in: Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research, SOSR '15, ACM, New York, NY, USA, 2015, pp. 9:1–9:7. doi:10.1145/2774993.2775003.
URL http://doi.acm.org/10.1145/2774993.2775003

[29] D. Hoffman, D. Prabhakar, P. Strooper, Testing Iptables, in: Proceedings of the 2003 Conference of the Centre for Advanced Studies on Collaborative Research, CASCON '03, IBM Press, 2003, pp. 80–91.
URL http://dl.acm.org/citation.cfm?id=961322.961337

[30] D. Hartmeier, Design and Performance of the OpenBSD Stateful Packet Filter (Pf), in: Proceedings of the FREENIX Track: 2002 USENIX Annual Technical Conference, USENIX Association, Berkeley, CA, USA, 2002, pp. 171–180.
URL http://dl.acm.org/citation.cfm?id=647056.713848

[31] S. Jouet, R. Cziva, D. P. Pezaros, Arbitrary packet matching in OpenFlow, in: 2015 IEEE 16th International Conference on High Performance Switching and Routing (HPSR), 2015, pp. 1–6. doi:10.1109/HPSR.2015.7483106.

[32] R. Fernando, S. Mackie, D. Rao, B. Rijsman, M. Napierala, M. Morin, Service chaining using virtual networks with bgp vpns, draft-ietf-bess-service-chaining-03 (July 2017).
URL https://tools.ietf.org/html/draft-ietf-bess-service-chaining-03