



UvA-DARE (Digital Academic Repository)

Evaluation of virtualization and traffic filtering methods for container networks

Makowski, Ł.; Grosso, P.

DOI

[10.1016/j.future.2018.08.012](https://doi.org/10.1016/j.future.2018.08.012)

Publication date

2019

Document Version

Final published version

Published in

Future Generation Computer Systems

License

Article 25fa Dutch Copyright Act (<https://www.openaccess.nl/en/policies/open-access-in-dutch-copyright-law-taverne-amendment>)

[Link to publication](#)

Citation for published version (APA):

Makowski, Ł., & Grosso, P. (2019). Evaluation of virtualization and traffic filtering methods for container networks. *Future Generation Computer Systems*, 93, 345-357.

<https://doi.org/10.1016/j.future.2018.08.012>

General rights

It is not permitted to download or to forward/distribute the text or part of it without the consent of the author(s) and/or copyright holder(s), other than for strictly personal, individual use, unless the work is under an open content license (like Creative Commons).

Disclaimer/Complaints regulations

If you believe that digital publication of certain material infringes any of your rights or (privacy) interests, please let the Library know, stating your reasons. In case of a legitimate complaint, the Library will make the material inaccessible and/or remove it from the website. Please Ask the Library: <https://uba.uva.nl/en/contact>, or a letter to: Library of the University of Amsterdam, Secretariat, P.O. Box 19185, 1000 GD Amsterdam, The Netherlands. You will be contacted as soon as possible.



Evaluation of virtualization and traffic filtering methods for container networks

Łukasz Makowski, Paola Grosso*

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

HIGHLIGHTS

- Digital marketplaces.
- Container addressing technology evaluation.
- Filtering evaluation in container networks.

ARTICLE INFO

Article history:

Received 31 January 2018

Received in revised form 16 June 2018

Accepted 7 August 2018

Available online 26 September 2018

Keywords:

Containers
Overlay networks
Packet filtering
Addressing
Multi-tenancy

ABSTRACT

Future distributed scientific applications will rely on containerization for data handling and processing. Likewise, emerging digital marketplaces will be built using these technologies. In particular, support for multi-tenancy will be a major requirement. The question is whether container networking is already mature enough to support the level of usability required in these environments.

With the work we present here we set out to experiment and evaluate three novel technologies that support addressing and filtering in container overlays: EVPN, ILA and Cilium/eBPF. We show that it is possible to integrate ILA and eBPF to support dynamic connections between containers. We describe performance testing of filtering operations with Cilium/eBPF and the initial demonstration of ILA showcased during the 2017 SuperComputing conference.

All the experiments presented in this article identify the basic building blocks for the much needed container networking multi-tenancy.

© 2018 Elsevier B.V. All rights reserved.

1. Introduction

Many business applications use containers to perform complex or distributed computational tasks [1]. The use of containers for scientific applications is also becoming more evident, in many fields from high energy physics to genomic research [2,3]. The motivation for such adoption lays in the flexibility and portability of this approach, compared to use of bare metal nodes or virtual machines. Low resources overhead, short startup time and the simplification of application deployment process all contribute to containers' widespread adoption.

This increased adoption goes hand in hand with new technical challenges for the operations of such distributed container systems. Some of these new challenges are intrinsic to the nature of scientific cooperation, which often relies on communication between scientists in different institutions.

An area that has started to get significant attention is container networking [4,5]. Networking containers together such that the

communication can occur across institutions requires the investigation and when possible adoption of emerging and developing technologies.

This ongoing development brings with it a number of shortcomings, which need to be addressed before containers can be heavily deployed by scientists in distributed settings. This is particularly true when we consider that both in industry and academia, we are experiencing an increased interest in data sharing platforms, aka *digital marketplaces*. Such platforms, which we will describe briefly in Section 2, plan to rely, at the infrastructural level, on containers. The urgency for stable, reliable and distributed container networks is therefore even more pressing.

A first challenge relates to addressing. Assuming a container uses the hosts' IP address for external 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 this requires the use of techniques such as port-level Network Address Translation (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

* Corresponding author.

E-mail addresses: makowski@uva.nl (Ł. Makowski), p.grosso@uva.nl (P. Grosso).

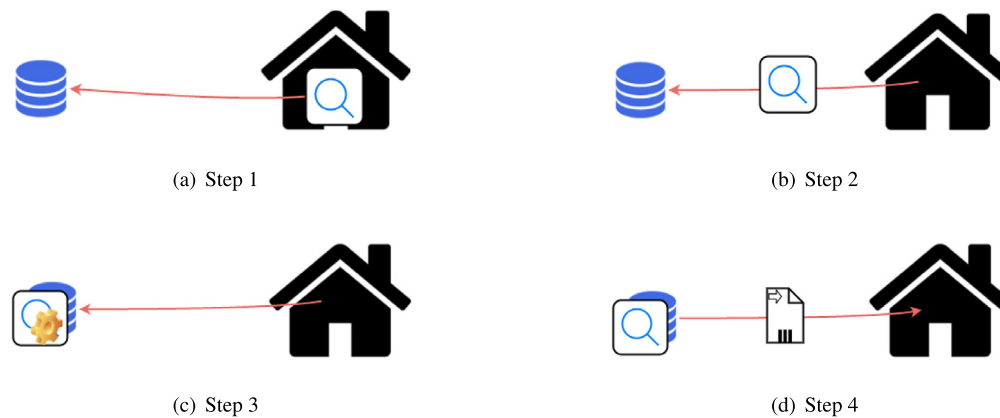


Fig. 1. Remote data-set analysis leveraging one container. Step 1: creating the search-container; Step 2: shipping a container to data location; Step 3: executing the search-container; Step 4: sending the results back.

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 capability to support a flexible environment for container systems. First, we analyzed 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 utilizing 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.

Finally, the integration of these technologies is paramount to support the distributed containers networks needed in science. In this paper, we present our work integrating ILA and eBPF, to support dataplane operations across distributed containers. That was demonstrated at the SuperComputing conference 2017 (SC17). This work is one of the basic building blocks in future container networks.

The paper is organized as follows, in Section 2 we briefly introduce the concept and vision behind digital marketplaces. After motivating the use of containers in such infrastructure we present in Section 3 the general features of container networks, and provide a more detailed introduction to EVPN, ILA and Cilium. We then describe our initial experiments to compare ILA and EVPN (Section 4). We continue with the more complex tests made for the implementation of ILA functionalities using eBPF (Section 5), followed by the performance evaluation of Cilium (Section 6). These two efforts resulted in our SC17 demonstration (Section 7). Finally, we present a summary of our findings (Section 8); and then compare our work with the related work in the area (Section 9). In Section 10 we identify future direction of work based on the current results.

2. Digital data marketplaces and containers

Data sharing and digital collaboration among businesses and scientists will become even more crucial in the coming years, thanks to their potential for increased efficiency, lower costs and lower pressure on infrastructure and environment [6]. These future data infrastructures will support data sharing among distributed partners; they will need to be easy to deploy, robust and easily adjustable to the participant needs.

A representative use of digital market relates to remote data processing. Data owners, either a company or a scientific institution, might have data that they need to share with others under certain policies. For example, scientific data-sets are usually made publicly available by universities, as well as private institutions; allowing the research community to freely use them for the purpose of analysis. However, some of these are not meant to be openly available due to licensing or privacy reasons. One of the typical limitations is that such data should not leave the organization's location. This significantly impacts the usability of such data-set for scientific purposes. To conduct the remote analysis, a person would need to prepare an application and request its execution by the remote party. In practice, this is hardly achievable due to, for example, incompatibilities of the used systems or the lack of manpower. In these cases, containers can be used to send *search functions* to the data and send back the filtered/retrieved information. Fig. 1 illustrates the basic building blocks of such approach: first, the creation of a container with the appropriate search function; then, the shipping of this container to the data location; followed by the actual search and finally the delivery of the results back.

This basic process can be generalized to the case of data sets residing at multiple locations, as illustrated in Fig. 2.

The outcome of a search executed by multiple containers might need to be post-processed e.g. unified so that a single response is returned back to the user who originated the processing. In order to realize this, there will be another container, i.e. *correlate-container*. This will implement the correlation function of choice, e.g. removing duplicate result entries. The correlate-container operates by receiving the output from N search-containers and combining those resulting in correlated output passed to the end user.

This last scenario is what we expect to be heavily adopted in digital marketplaces. This requires an operational network among all the containers involved, hence the need for reliable container overlay networks.

3. Container networking

Container networks are in essence network overlays. The concept of overlay networks is not a new idea, but in the current virtualisation era, it is getting more and more attention. The IETF's Network Virtualization Overlays (nvo3) working group produced a number of documents discussing design and architecture of network virtual overlays (NVOs). Lasserre et al. [7] defined the NVO fundamental functional components, which are illustrated in Fig. 3.

This reference model distinguishes between three elements:

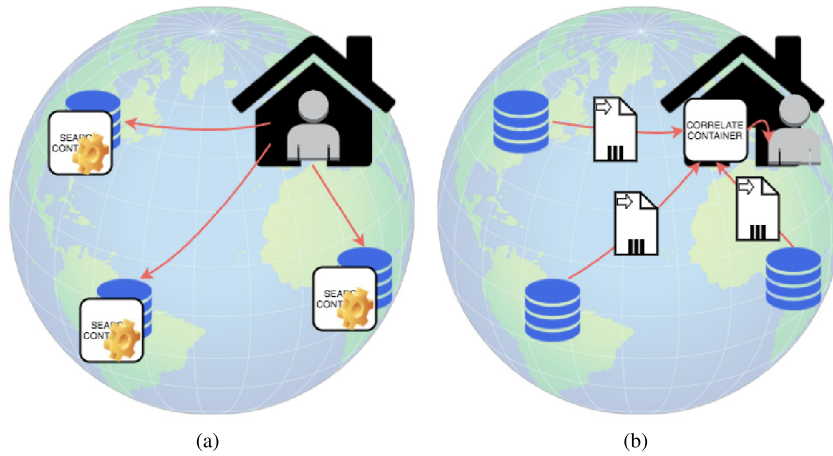


Fig. 2. Remote data-set analysis in multi-site scenario. The container is copied, delivered to N=3 remote locations and executed (a). N=3 distinct results are passed back to the correlate-container, which merges the results (b).

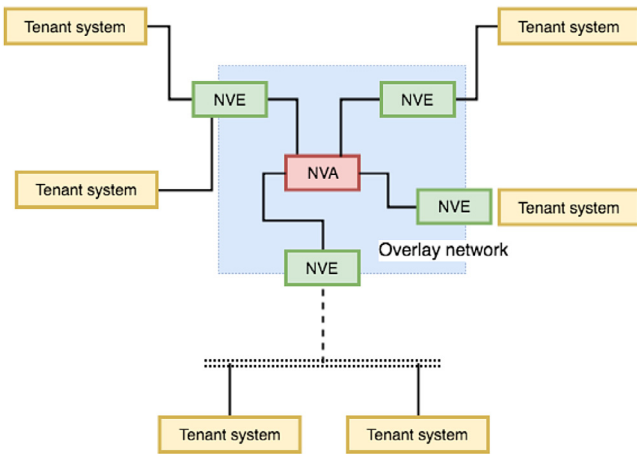


Fig. 3. NVO reference model.¹

- 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. [8] emphasize multi-tenancy as a major feature for modern NVOs. The important attribute of virtual network is the ability to isolate particular VN instances from each other, while being TS addressing agnostic.

Our focus is specifically on NVO built for the purpose of using containers. Important network functionalities that we expect to find in this environment are:

- Service function chaining (SFC) and load-balancing. A container network, as any network, requires a generic set of features such as load-balancing, firewalling etc. So that an NVO could natively support steering traffic through, without the need to modify application workflow.

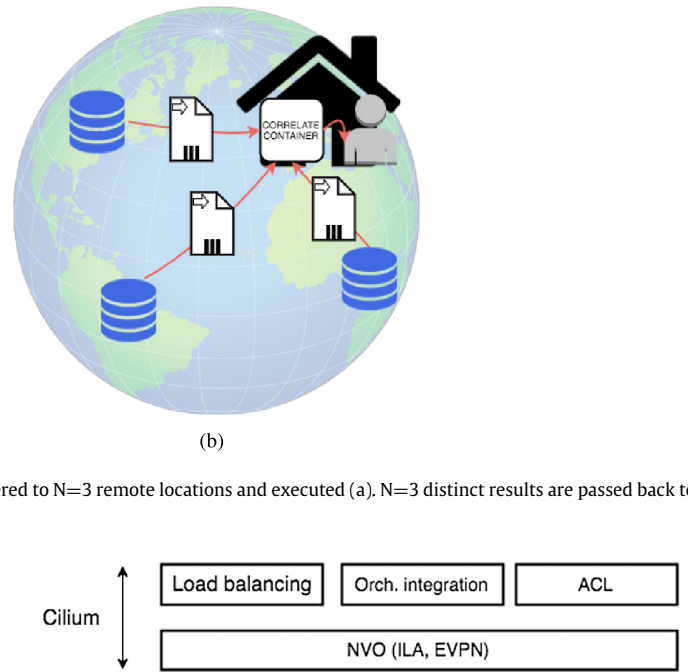


Fig. 4. Container network components. An NVO should have features such as load-balancing, orchestrator integration and access control lists (ACL).

- Traffic policies. Maintaining consistent and accurate filtering rules has always been a challenge for complex computer systems. Evidently, within a system which has a potential for spawning multiple endpoints across many servers, it only gets more significant.

Fig. 4 illustrates the container network components and features we discussed. It also shows how some of the technologies we have investigated fit in this schema: a number of them target specifically the creation and support of an NVO, e.g. ILA, EVPN with VXLAN; others such as Cilium support the whole set of features we discussed before.

In the upcoming sections we will provide a short overview of the three technologies we have investigated.

3.1. EVPN

EVPN is a mature standard which evolved from the family of MPLS based L2VPN [9], L3VPN [10] solutions for providing ISP-grade technology of L2 and L3 virtual private networks (VPNs) spanning multiple customer locations.

The concept of EVPN has been introduced and formalized in [11]. EVPN cuts down on its predecessors' concepts reducing its scope to Ethernet and IP protocols family only. This allows reduced complexity while still enabling the creation of scalable Ethernet networks. A notable change is that it permits the use of other packet encapsulation techniques. By introducing VXLAN [12] into EVPN, the latter has opened itself a way to the mid and low range devices not supporting MPLS format. Additionally, the solution is flexible enough to be used with other encapsulation standards as long as there is a way to embed a VNID into a packet. Hence there are existing EVPN variations (e.g. in OpenContrail²) employing MPLS-in-UDP [13] or MPLS-in-GRE [14]. The RFC draft [15] aims to be more specific and discusses the scenarios of how EVPN can be used to build an NVO network, going over available encapsulations, multi-homing scenarios and multicasting support.

¹ <https://tools.ietf.org/html/rfc7365>.

² http://www.opencontrail.org/opencontrail-architecture-documentation/#section1_4.

3.2. ILA

ILA is an NVO approach proposed by Herbert and Lapukhov [16]. It has been designed to satisfy two requirements:

- Unique addressing. ILA provides a unique IPv6 address for every container/task running in the overlay.
- Address mobility. In specific, when a network endpoint gets restarted on a different physical node, it is still able to maintain its virtual address.

ILA is inspired by other Identifier/Locator protocols such as ILNP [17] and LISP [18]. Nonetheless, its creators argue that, unlike ILNP, ILA is easier to deploy in existing infrastructures as it does not require new encapsulation and has immediate compatibility with existing network equipment.

Instead of encapsulation, ILA uses the left 64 bits of an IPv6 address as a Locator, identifying the physical node and the remaining 64 bit part as Identifier, to uniquely address an endpoint.

The endpoints in an ILA network (e.g. containers) are unaware of the Locator part of an address. While they still use IPv6 addresses to communicate, the ILA packet destination address is subject to a translation. From the 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. 5). By using this approach, endpoint mobility is supported as the address is not tied to an underlay and can be placed anywhere within ILA network.

The ILA authors also published an additional document focused on deployment approaches [19]. They discuss how to integrate ILA with multiple Linux kernel namespaces; this insight is very useful for us given our goal of using ILA as a container network. Although the ILA standard does not enforce any particular solution as NVA of choice, a recent draft document [20] identifies a number of required MP-BGP [21] extensions; this indicates that in the future ILA could follow the approach taken by EVPN.

3.3. Cilium

Cilium leverages eBPF as technology for traffic filtering and policy definition. The eBPF [22] Linux kernel mechanism has a potential to enhance a traditional way of packet filtering with the aid of netfilter [23,24]. The eBPF's predecessor – classical BPF (cBPF) [25] 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 [26] or DDoS protection [27].

Currently, a single eBPF program has a limitation of 4096 instructions, however, as it can tail-call another program, with a maximum of 32 times, this is not a restriction that significantly restricts functionality. Because eBPF code is stateless by design, a special construct (*maps*) has been created to allow saving information that can then 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 a maximum of 64 maps. In Fig. 6 we illustrate how an eBPF program can be used. The common way of creating eBPF programs is to use languages like C or P4. These programs can be compiled into the ELF³ format with the aid of LLVM.⁴ Once compiled, an eBPF program can be used in-kernel by using a *bpf()* system call. After the program passes the verification it might be attached to a hook point, e.g. the Linux traffic control (TC) sub-system.

Using the Linux traffic control (“tc”) as a point of attachment allows to perform actions on packets before they enter 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 a packet takes in the kernel. Cilium generates eBPF programs which are further hooked to the tc sub-system, resulting 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 that represents a group of containers. A network operator may decide to specify rules (*policies*) on how these groups, identified by their own unique labels, can communicate. Such rule is automatically assigned with an ID called an identity, which is allocated to any container affected by it (Listing 2).

```
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 implementation, Cilium uses VNID field to associate a network packet with the 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 set identity field.

4. ILA and EVPN comparison

The first step of our work has been a comparison of ILA and EVPN based on the documentation available (see Section 4.1). The following step has been the creation of two distinct test environments (see Sections 4.2 and 4.3) to assess the ease of use of the two technologies (see Section 4.4).

4.1. Multi-tenancy comparison

Our initial focus has been on analyzing the support for multi-tenancy of the two technologies. We looked at the type of networks that can be realized; how the data plane and the control plane are implemented; last, we compared how multiple virtual networks (VNs) are supported. Table 1 summarizes the results of this initial evaluation.

³ <http://wiki.osdev.org/ELF6>.

⁴ <https://llvm.org/>.

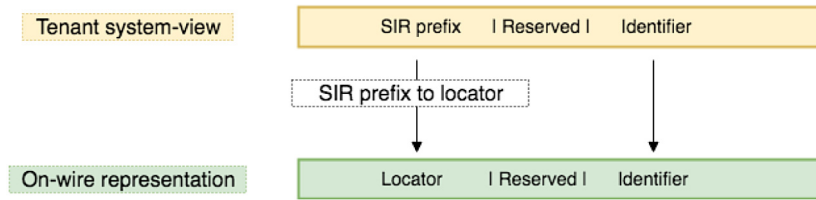


Fig. 5. ILA address translation overview. A SIR prefix in packet’s destination address is replaced with a Locator before it is transmitted across an NVO.

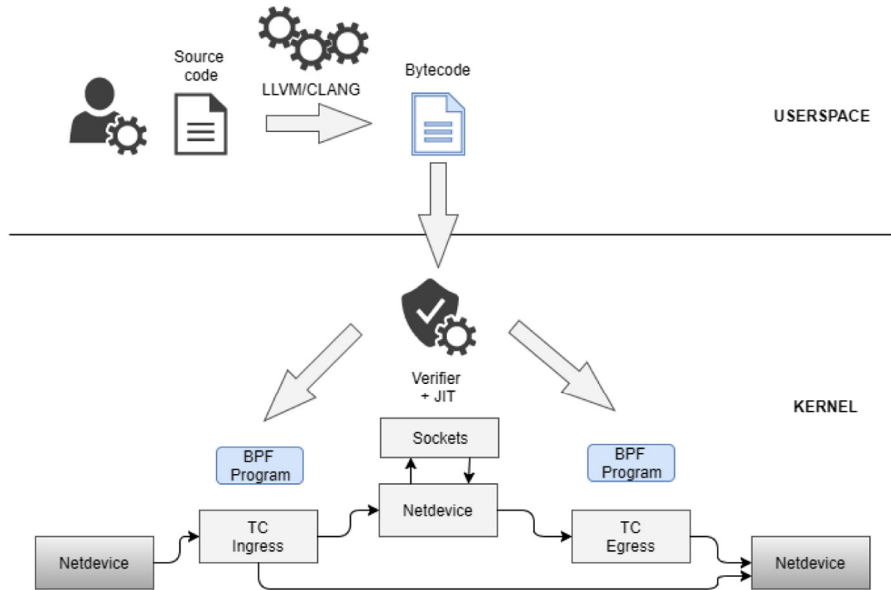


Fig. 6. The overview of eBPF program attachment.⁵

Table 1
EVPN-ILA feature comparison.

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 ^a	SIR ^b , VNID

^aRoute Target/Route Distinguisher.
^bStandard Identifier Representation.

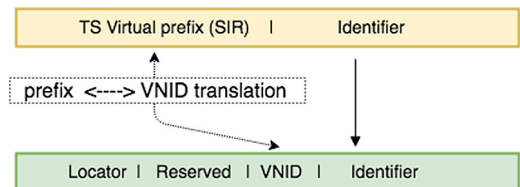


Fig. 7. ILA multi-tenancy concept. For every packet the VNID can be translated to a tenant virtual prefix, and vice-versa.

The last feature in the table, VN context, is crucial to support multiple virtual networks. This requires packets to provide information, in the form of a VNID, regarding which context they belong to.

The size of the VNID space assumed to meet the majority of demands is one million values (20 bits) [28]. Both, EVPN with VXLAN and ILA fulfill this requirement. In the EVPN with VXLAN case this is quite straightforward as VXLAN uses its dedicated packet field (20 bits) for that purpose. In the case of ILA this is more complex. ILA can devote 28 bits of the Identifier part for the VNID, which can, in turn, be used for the translation between the ILA’s tenant virtual prefix and the on-the-wire representation (see Fig. 7).

There is though a difference between ILA and EVPN when it comes to the choice of the VNID. In ILA it is possible to host multiple tenants and to isolate the traffic in a scope of a single VNID; however, the address assignment must be compliant with the ILA metadata fields. 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.

4.2. EVPN test environment

To prototype an EVPN network we used the components and the general idea described in [29].

As shown in Fig. 8, the topology consisted of two VMs operating as container hosts and an additional one running as a route-server. Each container host was running a BGP daemon (bagpipe-bgp⁶) peering with a route-server (GoBGP⁷). 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

⁵ <https://cilium.readthedocs.io/en/latest/architecture/>.

⁶ <https://github.com/Orange-OpenSource/bagpipe-bgp>.

⁷ <https://osrg.github.io/gobgp/>.

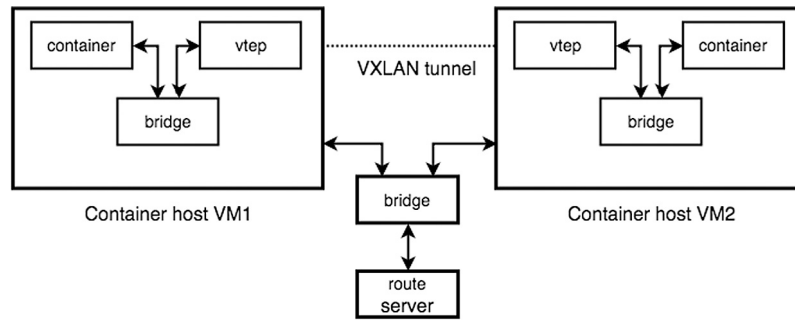


Fig. 8. EVPN environment topology.

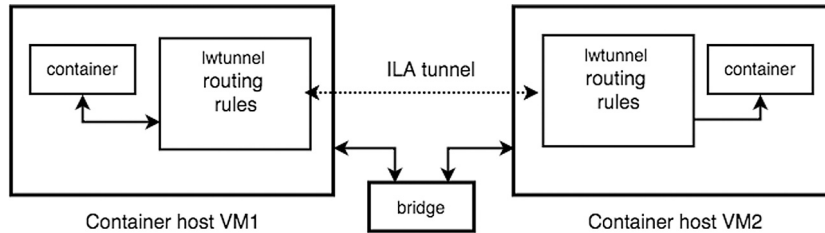


Fig. 9. Test topology.

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.

4.3. ILA test environment

The ILA environment (Fig. 9) 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)

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 the 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` option. 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).

4.4. Ease-of-use

Our efforts to create two working environments for EVPN and ILA provided us with a good insight into 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 allowed

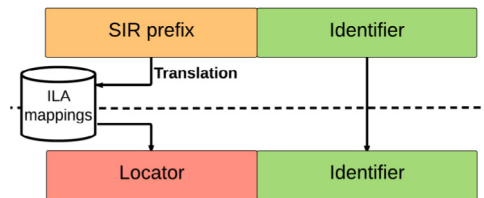


Fig. 10. The ILA eBPF translation. Upon packet’s arrival, `ila.c` consults the eBPF map holding ILA mappings. If the map lookup is successful (i.e. there was a key corresponding to SIR+Identifier address), the SIR prefix gets swapped with the Locator.

us to concentrate our focus on configuring BGP sessions, as well as on the actual components integration.

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 [30]. 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 necessity to manually create ILA tunnels between the containers.

5. ILA eBPF implementation

After evaluating the ILA Linux kernel module, we decided to implement ILA data-plane operations with the aid of eBPF. The motivation for that was two-fold: (1) a better understanding of ILA’s specifics, (2) gaining the potential of having higher packet rewrite performance.

```

Border Gateway Protocol - UPDATE Message
2   Marker: ffffffffffffffffffffffffffffffff
   Length: 107
4   Type: UPDATE Message (2)
   Withdrawn Routes Length: 0
6   Total Path Attribute Length: 84
   Path attributes
8   Path Attribute - ORIGIN: IGP
   Path Attribute - AS_PATH: empty
10  Path Attribute - LOCAL_PREF: 100
   Path Attribute - EXTENDED_COMMUNITIES
12  Path Attribute - MP_REACH_NLRI
   Flags: 0x80, Optional: Optional, Non-transitive, Complete
14  Type Code: MP_REACH_NLRI (14)
   Length: 48
16  Address family identifier (AFI): Layer-2 VPN (25)
   Subsequent address family identifier (SAFI): EVPN (70)
18  Next hop network address (4 bytes)
   Number of Subnetwork points of attachment (SNPA): 0
20  Network layer reachability information (39 bytes)
   EVPN NLRI: MAC Advertisement Route
22  AFI: MAC Advertisement Route (2)
   Length: 37
24  Route Distinguisher: 0001c0a8320b0094 (192.168.50.11:148)
   ESI: 00 00 00 00 00 00 00 00
26  Ethernet Tag ID: 0
   MAC Address Length: 48
28  MAC Address: 52:da:46:13:7e:7d (52:da:46:13:7e:7d)
   IP Address Length: 32
30  IPv4 address: 10.1.2.8
   MPLS Label Stack: 136 (bottom)

```

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

```

ip -6 a 1
#(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

```

#egress route
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. The ILA tunneling routes. The first one is handling incoming packets' translation and the second route translates the outgoing packets.

We created the `ila.c` eBPF program, which was divided in two BPF code sections⁸:

- `loc_if`, handling ingress ILA-attached container traffic
- `cont_if`, processing packets on egress of an ILA-attached container

⁸ An ELF section which contains a set of eBPF operations. See `man tc-bpf`.

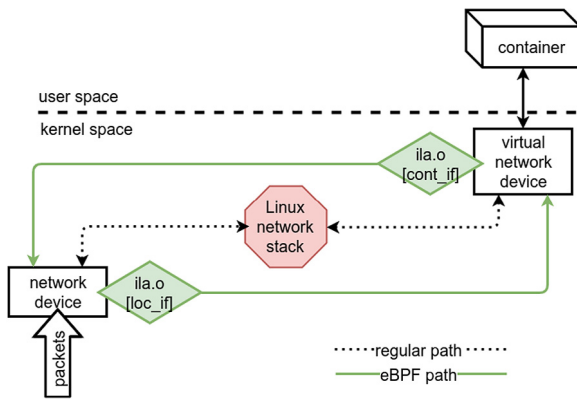


Fig. 11. The `ila.o` attachment outline. Section `loc_if` is attached to the container-host network device, whereas `cont_if` is hooked to the veth interface leading towards the container.

Next, an eBPF map (BPF_MAP_TYPE_HASH⁹ type) was defined and accessed by both code sections. As illustrated in Fig. 10, its role was to hold the ILA mapping information allowing the program to look for a matching key (packet's destination address) and rewrite it with looked up corresponding value (the SIR prefix – on ingress and the Locator on egress).

In order to put an eBPF program into actual use this has to be compiled into an object file. Once done, the program (in our case the `ila.o` file) needs to be applied to the network interface it is supposed to operate on.

In Fig. 11, we illustrate where the particular code sections of our program were attached, and what was the path taken by packets once those were in place. In our solution we were using the interface's `tc clsact` queuing discipline. In Fig. 6 we had present the example commands we used.

5.1. eBPF internals

In Fig. 7 we present the dump of `/sys/fs/bpf/tc/global-s/OUT_MAP` map used in our ILA program. Going over the first listed entry it can be seen that the key (`deadbeef 00 00 00 00 00 00 00 00 00 00 11`) is actually the ILA address consisting of SIR (`deadbeef : /64`) and Identifier (`: : 11/64`). Whereas, the value (`10 c1 00 00 00 00 00 00`) is actually the 64-bit long Locator address.

To illustrate how our program was using the map information, in Fig. 8 we present the code of `set_tcp_ip6_dst` function we used. On line 4, the packet's current IPv6 destination address is read (`bpf_skb_load_bytes` function) and saved under the `old_prefix` variable. In the following line, the function `bpf_skb_load_bytes` looks up a value corresponding to `old_prefix` variable. On line 11, if the lookup is successful, the function `bpf_skb_store_bytes` rewrites packet's address with a new value.

5.2. ILA control-plane and mappings

The crucial part of this effort was the creation of a control-plane that would distribute ILA mapping information across all container-hosts participating in the overlay. Usecases such as the ones described in Section 2 rely heavily on this type of functionality.

Specifically, if a new container is deployed, a new set of ILA mappings must be installed locally, as well as at other container-hosts in the topology.

In order to achieve this, we deployed the distributed Consul¹⁰ key-value database, where each container-host run one KV store node. In Fig. 12 we show how all elements were coupled together. For brevity, we illustrate only the case with two container-hosts, showing all key elements where a packet from container-1 to container-2 is sent.

Once the containers are started and assigned with their ILA addresses, the corresponding mapping rules are generated and installed locally. The special sync process watches the eBPF map and pushes its content to the local KV store node. Next, the new information gets replicated among all KV store nodes. Looking from the perspective of container-host-1, there are now new mappings present about the container started at a remote host (container-host-2). The previously mentioned sync process, fetches the new entries from Consul and updates the eBPF map to reflect them. The end result is that the same eBPF map entries are residing at both container hosts; this effectively makes them ready to relay the ILA traffic between the containers.

When container-1 decides to contact container-2, it will create and transmit a packet addressed to recipient's ILA address i.e. `dead:beef::2000:0000:0000:0002`. Next, the packet will be intercepted by `ila.o` program (`cont_if` section), which will inspect if there is a mapping information present. If that is the case, the program will rewrite the destination address to `10c2::2000:0000:0000:0002` (container-host-2's Locator+Identifier), and send the packet over its default interface. Once the packet arrives on the container-host-2, it gets checked by `loc_if` `ila.o`'s section. If the mapping corresponding to the destination address exists, the program rewrites it to `dead:beef::2000:0000:0000:0002` (notice the address is now the same as generated by container-1) and pushes it over veth link leading directly towards the container-2.

6. 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. 13) 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 the following 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

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 Cilium's traffic policies.

⁹ See `man bpf`.

¹⁰ <https://www.consul.io/>.

```
tc filter add dev enp0s8 ingress prio 1 handle 1 bpf da obj ila.o sec loc_if
tc filter add dev veth5e3c0cf ingress prio 1 handle 1 bpf da obj ila.o sec cont_if
```

Listing 6. The ila.o attachment sample commands. The first one attaches a loc_if code section to a container-host interface. In the next line, section cont_if is plugged into a container-facing veth device.

```
bpf-map d /sys/fs/bpf/tc/globals/OUT_MAP | grep .
Key:
00000000 de ad be ef 00 00 00 00 00 00 00 00 00 00 11 |.....|
Value:
00000000 10 c1 00 00 00 00 00 00 |.....|
Key:
00000000 10 c1 00 00 00 00 00 00 00 00 00 00 00 11 |.....|
Value:
00000000 de ad be ef 00 00 00 00 |.....|
Key:
00000000 de ad be ef 00 00 00 00 00 00 00 00 00 12 |.....|
Value:
00000000 10 c2 00 00 00 00 00 00 |.....|
Key:
00000000 10 c2 00 00 00 00 00 00 00 00 00 00 00 12 |.....|
Value:
00000000 de ad be ef 00 00 00 00 |.....|
```

Listing 7. A sample eBPF map content used by ila.c program

```
1 static inline int set_tcp_ip6_dst(struct __sk_buff *skb)
2 {
3 struct in6_addr old_prefix;
4 bpf_skb_load_bytes(skb, IP6_DST_OFF, &old_prefix, sizeof(struct in6_addr));
5
6 __u64 * vp = 0;
7 vp = bpf_map_lookup_elem(&OUT_MAP, &old_prefix);
8
9 if(vp){
10 __u64 v = *vp;
11 uint32_t ret = bpf_skb_store_bytes(skb, IP6_DST_OFF, &v, sizeof(v), 0);
12 return ret;
13 }
14 else{
15 return -1;
16 }
17 }
```

Listing 8. The body of set_tcp_ip6_dst function. The function is used when a packet is classified as the IPv6 type and has TCP used for the transport layer.

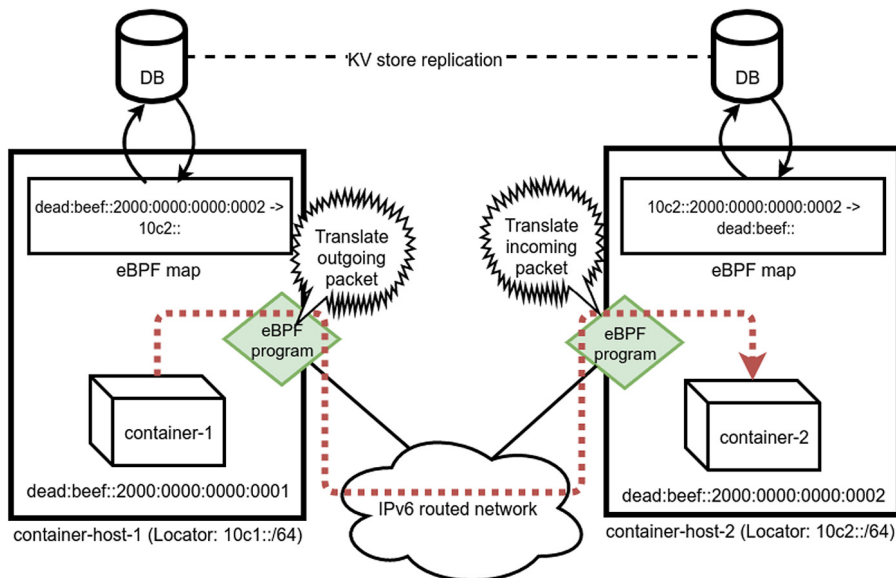


Fig. 12. ILA eBPF operations overview for a packet sent from container-1 to container-2.

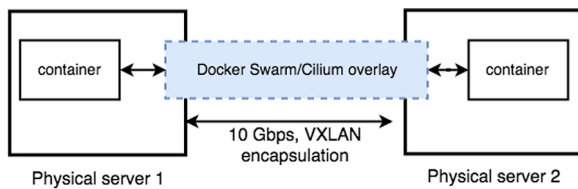


Fig. 13. Netfilter/Cilium performance test topology.

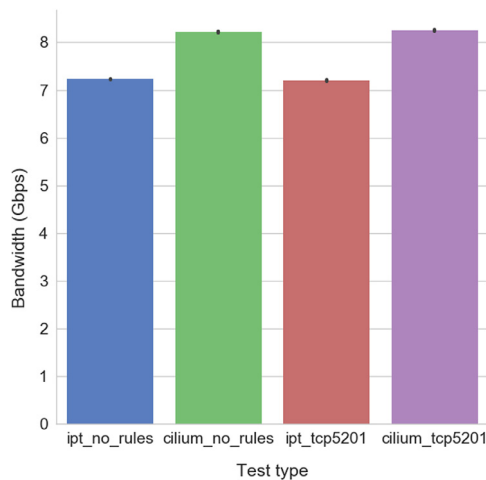


Fig. 14. Iptables and Cilium based test filtering comparison. Two leftmost columns show the results without filtering rules, the next two depict the outcome with the filter applied. The bars represent mean value calculated over $N=6000$ samples. The error bars generated for $CI=95\%$ are an order of magnitude smaller, thus they have a form of black dots residing on top of the bars.

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 9 we present the way we implemented this with netfilter, whereas Listing 10 illustrates Cilium's policy configuration.

In total, we performed 4 different types of measurements. The common element was that we performed a hundred 70 second-long "iperf3" TCP bandwidth measurements between two containers placed on distinct hosts. Each "iperf3" run reported the bandwidth in one-second intervals where the first ten measurements were omitted. This resulted in $N=6000$ collected samples per experiment.

6.1. Filtering comparison

In our experiments we measured the bandwidth speed of a TCP stream between two containers interconnected with a network overlay. The results of conducted performance tests, including the errors for the 95% CI, are depicted in Fig. 14. There are four bars in total, each for one type of the test:

- 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

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. 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.

7. SC17 demo

We presented our work during 4th International Workshop on Innovating the Network for Data Intensive Science (INDIS), which took place during the SC17 conference in Denver. For this purpose, we designed and showcased an ILA overlay demonstration for the Exhibition at SC17; the demo was based on the efforts described in Section 5.

Our demo illustrated the usecase of an overlay spanning among four containers residing in separate locations. Our goal was to show how ILA can be the technology of choice when building such an overlay. Users of our demo could initiate the communication among pairs of containers of their choice. The user was presented with the map where the locations of containers were indicated (Fig. 15). After clicking on a corresponding icon, a container shell session was spawned allowing to execute commands. The user could observe the ILA translation process and ultimately the successful ILA communication among the chosen overlay components.

Fig. 16 shows the demo's architecture. The setup consisted of four virtual machines, each representing the different locations shown in the UI (New Orleans, Austin, Salt Lake City, Denver). The VMs were fulfilling the role of container hosts, being preinstalled with a community release of Docker container engine. In each container-host there were five components present:

- container, acting as endpoints in the ILA overlay
- eBPF program and its map (holding ILA mappings)
- BIRD¹¹ BGP daemon
- Consul database node

The BGP daemon was needed to provide routes towards other ILA Locator prefixes assigned to each container-host. The corresponding nodes were interconnected in an iBGP full-mesh manner.

8. Discussion

EVPN and ILA evaluation

The EVPN and ILA environments we created were capable of performing basic operations; however, as our work shows both technologies are still in preliminary stages. Crucially, both setups, EVPN and ILA, lack the integration with a container orchestrator that would allow to dynamically span virtual networks according to the deployed setup.

When focusing on container-based workloads, such as the ones we envision for our digital marketplaces, we deemed EVPN to be less suitable. The fact that EVPN carries the original packet inside VXLAN is in our opinion a redundant overhead.

ILA, despite its immaturity, seems a better fit for providing the Layer 3 networks between containers that we need. Even though ILA requires certain parts of the address to be fixed (multi-tenancy, reserved bits) or adjusted (checksum-neutral mapping), we do not consider this to be particularly problematic for a container-based system. In those environments, the auxiliary utilities providing the means to discover services are common and therefore the need to assign a specific address is less relevant. Still, we did not address all the issues related to the adoption of ILA. Even though ILA

¹¹ <http://bird.network.cz/>.

```

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 9. Netfilter filtering policy applied within the namespace of the overlay network (1-s8idcnjdiq) created by Docker Swarm.

```

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

```

Listing 10. Cilium filtering policy JSON file.

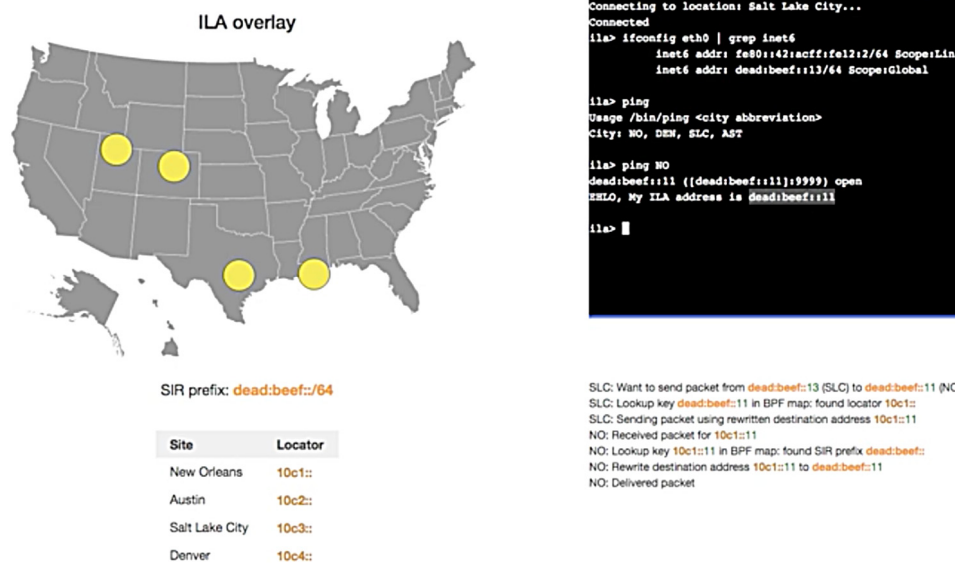


Fig. 15. The UI interface of our ILA demo. The map on the left shows the container locations (yellow dots). By clicking on a location a terminal session to a container is spawned, this allows testing the communication to the other containers in the overlay. After initiating the communication, the subsequent steps of the ILA translation process are displayed in the lower right part of the UI.

does not create overhead due to encapsulation, it has implications related to the IPv6 address rewrites. Given that after the translation an address is changed, the checksum of the corresponding TCP segment is effectively invalid. To mitigate the possibility of such packets getting discarded, the ILA authors proposed checksum-neutral mapping mechanism [16], however we did not experiment with this approach.

Moreover, we did not yet implement the VNID encoding needed for multi-tenancy. This is one of the key requirements needed by the digital data marketplace environments and will be the focus of our future work.

Cilium performance advantage

As our results showed in Section 6.1 the eBPF-accelerated Cilium demonstrates noticeably higher bandwidth than the overlay setup based on Docker Swarm. Interestingly, the filtering capabilities of the former are not playing the key role in those results. We

believe that the fact Cilium implements the packet's processing path mainly in eBPF, as we described in Section 3.3, 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. By doing this we avoided the need for per-packet sequential rule evaluation, which it is 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 corner cases which we did not encounter with our design.

9. Related work

Del Piccolo et al. [31] conducted a comprehensive study on the available solutions enabling mobility and multi-tenancy in

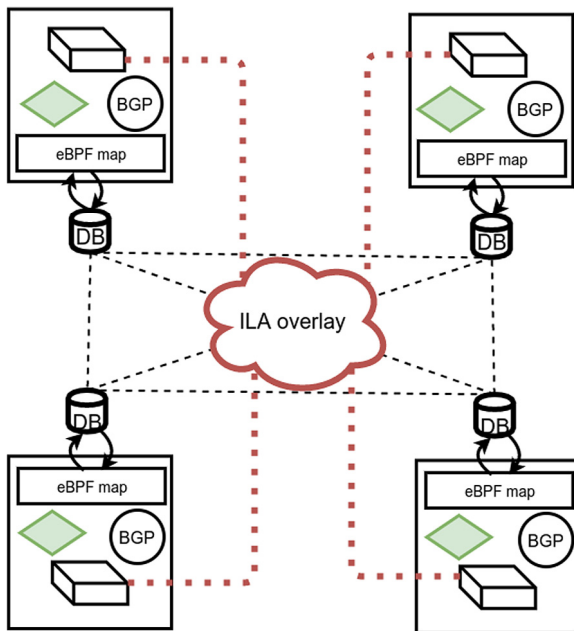


Fig. 16. SC17 demo architecture. Four containers (indicated as cubes) were interconnected with ILA overlay. The solution's backend consisted of a distributed database holding the ILA mappings; the eBPF `ila.o` program (green lozenge) and the BGP daemon. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

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. [32] propose a new overlay approach called *NoEncap*. The authors discuss 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 have been studied [33,34], there is currently no evaluation performed in the context of overlay networks for containers, comparable to the one we provided in our paper. Furthermore, even if Jouet et al. [35] used BPF to enhance on the packet matching in OpenFlow, we considered more instructive and worthwhile to provide a side-by-side comparison of netfilter and eBPF when they are applied to identical workloads.

10. Conclusions and future work

The adoption of containers for distributed and multi-domain scientific applications will rely on the creation of suitable container

networks. This will, in turn, require to solve problems related to addressing and filtering that emerge in these multi-tenant environments.

The work presented in this article shows a thorough evaluation of a number of developing 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 conclusions that are interesting for the networking community looking to deploy containers networks for support of science use cases.

First of all, we find EVPN to be more flexible regarding multi-tenancy than ILA. The former can host endpoints with conflicting addressing spaces without any constraints, whereas ILA encodes metadata in the Locator part of an original packet's address; this effectively puts extra limitations on the used IPv6 addresses.

Furthermore, we find EVPN to be a more mature solution than ILA. Data and control plane options defined by the EVPN standard are commonly available; this significantly shortens the time required to bootstrap. Whereas, the ILA has only proof-of-concept data-plane implementations and requires more development effort. Nevertheless, ILA holds a couple of attributes which make it particularly appealing for a container based environment. For example, the lack of encapsulation overhead is beneficial for avoiding issues with the incompatible intermediate devices as well as saving goodput for overlays not requiring Ethernet-level communication. As such, if used to empower digital data marketplaces, ILA would offer a lightweight way of creating virtual networks spanning across more than one administrative domain.

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

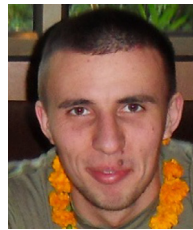
Acknowledgments

This work is funded by a grant from SURFnet in the program Research on Networks (2017), The Netherlands. Special thanks go to Ronald van der Pol and Marijke Kaat for their support during this work. Secondly, we would like to thank our colleague Ben de Graaff for the support and sharing his experience in regard to eBPF programming, as well as, his contribution to the development of ILA SC17 demonstration. We are also indebted to Tako Mars and Nick de Bruijn that paved the way for this research during their master graduation projects.

References

- [1] C. Pahl, Containerization and the paas cloud, *IEEE Cloud Comput.* 2 (3) (2015) 24–31.
- [2] D. Bonacorsi, G. Eulisse, T. Boccali, E. Mazzoni, Containerization of CMS applications with Docker, *PoS* (2016) 007.
- [3] W.L. Schulz, T.J. Durant, A.J. Siddon, R. Torres, Use of application containers and workflows for genomic data analysis, *J. Pathol. Inf.* 7 (2016).
- [4] 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.
- [5] 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.
- [6] A. Deljoo, T. van Engers, L. Gommans, C. de Laat, A normative agent-based model for sharing data in secure trustworthy digital market places, in: *The 10th International Conference on Agents and Artificial Intelligence, ICAART'18*, 2018.
- [7] M. Lasserre, F. Balus, T. Morin, N. Bitar, Y. Rekhter, Framework for Data Center (DC) Network Virtualization RFC7365, 2014. URL <http://tools.ietf.org/rfc/rfc7365.txt>.
- [8] T. Narten, E. Gray, D. Black, L. Fang, L. Kreeger, M. Napierala, Problem Statement: Overlays for Network Virtualization RFC7364, 2014. URL <http://tools.ietf.org/rfc/rfc7364.txt>.
- [9] K. Kompella, Virtual Private LAN Service (VPLS) Using BGP for Auto-Discovery and Signaling RFC4761, 2007. URL <https://tools.ietf.org/html/rfc4761>.

- [10] E. Rosen, BGP/MPLS IP Virtual Private Networks (VPNs) RFC4364, 2006. URL <https://tools.ietf.org/html/rfc4364>.
- [11] A. Sajjasi, BGP MPLS-Based Ethernet VPN, RFC7432, 2015. URL <https://tools.ietf.org/html/rfc7432>.
- [12] M. Mahalingham, Virtual eXtensible Local Area Network (VXLAN): A Framework for Overlaying Virtualized Layer 2 Networks over Layer 3 Networks, RFC7348, 2014. URL <https://tools.ietf.org/html/rfc7348>.
- [13] X. Xu, N. Sheth, L. Yong, R. Callon, D. Black, Encapsulating MPLS in UDP, RFC7510, 2015. URL <http://tools.ietf.org/rfc/rfc7510.txt>.
- [14] T. Worster, Y. Rekhter, E. Rosen, Encapsulating MPLS in IP or Generic Routing Encapsulation (GRE) RFC4023, 2005. URL <http://tools.ietf.org/rfc/rfc4023.txt>.
- [15] A. Sajjasi, A Network Virtualization Overlay Solution using EVPN, draft-ietf-bess-evpn-overlay-08, 2017. URL <https://tools.ietf.org/html/draft-ietf-bess-evpn-overlay-08>.
- [16] T. Herbert, Identifier-locator addressing for IPv6, draft-herbert-nvo3-ila-04, 2017. URL <https://tools.ietf.org/html/draft-herbert-nvo3-ila-04>.
- [17] R. Atkinson, Identifier-Locator Network Protocol (ILNP) Architectural Description, 2012. URL <https://tools.ietf.org/html/rfc6740>.
- [18] D. Farinacci, The Locator/ID Separation Protocol (LISP), RFC6742, 2013. URL <https://tools.ietf.org/html/rfc6742>.
- [19] P. Lapukhov, Deploying Identifier-Locator Addressing (ILA) in datacenter networks, Internet Engineering Task Force, draft-lapukhov-ila-deployment-01, Work in Progress, 2016. URL <https://datatracker.ietf.org/doc/html/draft-lapukhov-ila-deployment-01>.
- [20] P. Lapukhov, Use of BGP for dissemination of ILA mapping information, draft-lapukhov-bgp-ila-afi-02, 2016. URL <https://tools.ietf.org/html/draft-lapukhov-bgp-ila-afi-02>.
- [21] T. Bates, Multiprotocol Extensions for BGP-4, RFC4760, 2007. URL <https://tools.ietf.org/html/rfc4760>.
- [22] A. Starovoitov, The Berkeley packet filter, <https://www.kernel.org/doc/Documentation/networking/filter.txt>.
- [23] P. Ayuso, Netfilter's connection tracking system, LOGIN: USENIX Mag. 31 (3) (2006).
- [24] P. Russel, H. Welte, Netfilter Hacking How-to, URL <https://www.netfilter.org/documentation/HOWTO/netfilter-hacking-HOWTO.txt>.
- [25] S. McCanne, V. Jacobson, The BSD Packet Filter: A New Architecture for User-level Packet Capture, 1992, Visited on: 07-06-2017. <http://www.tcpdump.org/papers/bpf-usenix93.pdf>.
- [26] IOVisor community, BPF-based Linux IO analysis, networking, monitoring, URL <https://github.com/iovisor/bcc>.
- [27] G. Bertin, XDP in practice: integrating XDP into our DDoS mitigation pipeline.
- [28] D. Black, J. Hudson, L. Kreeger, M. Lasserre, T. Narten, An Architecture for Data-Center Network Virtualization over Layer 3 (NVO3), 2016. URL <http://tools.ietf.org/rfc/rfc8014.txt>.
- [29] M. Mukhtarov, Experiments with container networking: Part 2, 2016. URL <http://murat1985.github.io/kubernetes/cni/2016/05/15/bagpipe-gobgp.html>.
- [30] P. Lapukhov, Internet-scale Virtual Networking using ILA, 2016, retrieved on: 05-06-2017. URL https://www.nanog.org/sites/default/files/20161018_Lapukhov_Internet-Scale_Virtual_Networking_v1.pdf.
- [31] V.D. Piccolo, A. Amamou, K. Haddadou, G. Pujolle, A survey of network isolation solutions for multi-tenant data centers, IEEE Commun. Surv. Tutor. 18 (4) (2016) 2787–2821, <http://dx.doi.org/10.1109/COMST.2016.2556979>.
- [32] 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, ISBN: 978-1-4503-3451-8, 2015, pp. 9:1–9:7, <http://dx.doi.org/10.1145/2774993.2775003>.
- [33] 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, <http://dl.acm.org/citation.cfm?id=961322.961337>.
- [34] 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, ISBN: 1-880446-01-4, 2002, pp. 171–180, <http://dl.acm.org/citation.cfm?id=647056.713848>.
- [35] S. Jouet, R. Cziva, D.P. Pazaros, Arbitrary packet matching in OpenFlow, in: 2015 IEEE 16th International Conference on High Performance Switching and Routing (HPSR), 2015, pp. 1–6, <http://dx.doi.org/10.1109/HPSR.2015.7483106>.



Łukasz Makowski works as a system and network engineer at SNE research group (University of Amsterdam). His duties are design, implementation, and administration of systems created as a part of group's activities. He holds an Engineer's degree in Applied Computer Science from AGH University of Science and Technology, Cracow and M.Sc. in System and Network Engineering from the University of Amsterdam. Łukasz's research interests focus on SDN, data center networks, and system virtualization technologies.



Dr. Paola Grosso is associate professor in the System and Network Engineering (SNE) group at the University of Amsterdam. She is the coordinator and lead researcher of all the group activities in the field of multi-scale networks and systems. Her research interests lie in the creation of sustainable e-Infrastructures, relying on the provisioning and design of programmable networks. She co-chaired the NML-WG (Network Markup Language Working Group) within OGF (Open Grid Forum). She been a member of the SC organizing committee for SCinet for 7 years. She has been involved in several FP-7 projects, namely NOVI, ENVRI, Geysers and MOTE. She currently participates in several national projects, such as SARNET, DL4LD, SecConNet and in EU H2020-funded projects such as FED4FIRE+, GN4 and ENVRIPLUS.