Container-based Network Function Virtualization for Software-Defined Networks

Richard Cziva, Simon Jouet, Kyle J. S. White and Dimitrios P. Pezaros
School of Computing Science, University of Glasgow, Glasgow, G12 8QQ, Scotland
{r.cziva.1, s.jouet.1, k.white.3}@research.gla.ac.uk, dimitrios.pezaros@glasgow.ac.uk
http://glanf.dcs.gla.ac.uk

***Abstract*—Today's enterprise networks almost ubiquitously deploy middlebox services to improve in-network security and performance. Although virtualization of middleboxes attracts a significant attention, studies show that such implementations are still proprietary and deployed in a static manner at the boundaries of organisations, hindering open innovation.**

In this paper, we present an open framework to create, deploy and manage virtual network functions (NF)s in OpenFlow-enabled networks. We exploit container-based NFs to achieve low performance overhead, fast deployment and high reusability missing from today's NFV deployments. Through an SDN northbound API, NFs can be instantiated, traffic can be steered through the desired policy chain and applications can raise notifications. We demonstrate the systems operation through the development of exemplar NFs from common Operating System utility binaries, and we show that container-based NFV improves function instantiation time by up to 68% over existing hypervisor-based alternatives, and scales to one hundred co-located NFs while incurring sub-millisecond latency.

I. Introduction

Enterprise networks rely on a wide spectrum of hardware-based network appliances or 'middleboxes' to transform, inspect, filter or otherwise manipulate network traffic on top of packet forwarding. In recent years, middleboxes have become fundamental parts of operational networks, providing approx. 45% of the network devices to enforce security (e.g., firewalls and intrusion detection) and performance (e.g., rate limiters, proxies, load-balancers) policies throughout the topology [1]. Recent studies have shown that the advent of diverse consumer devices that rely on different, network-intensive cloud services as well as the increasing need of in-network security will increase the demand for middleboxes even further [2]. However, despite their increasing popularity, hardware-based middleboxes have significant drawbacks: they incur significant capital investment due to being provisioned and optimized for peak-demand, are cumbersome to maintain due to the expert knowledge required, and cannot typically be extended to accommodate new functionality as new operational requirements emerge. The proprietary software on which they run, limits innovation and creates vendor lock-in [3].

Network Function Virtualization (NFV) is a novel approach to address the above shortcomings of managing closed and proprietary appliances by decoupling network functions (NF)s from their hosting hardware platform. By using low-cost commodity servers, NFV can reduce Capital and Operational Expenditure and maximize Return on Investment (RoI) [4]. However, current early-stage deployments and platforms of NFV by large ISPs and DC network operators suffer from the statically-configured underlying routing mechanisms in place that do not support open interfaces and result in operator and environment-specific solutions in static or semi-static environments [5] [6]. For example, deploying one or more network functions requires the update of all affected switches' routing tables to redirect traffic, therefore making it impractical to deploy infrastructure-wide NFs. Consequently, current NFV platforms exhibit poor component reuse, and are still unable to fulfill dynamic, temporal traffic workloads in an elastic manner [7] [8]. In such environments, there is no cross-layer information exchange between the routing layer and the network functions, which results in a limited view of the network to each functional entity. We argue that improvements in NFV can be achieved by synergistic management and optimisation of NFs and end-to-end routing between hosts and NFs.

At the same time, Software-Defined Networking (SDN) has emerged to logically centralise the network's control plane with OpenFlow as its leading realization [9] [10]. SDN is penetrating in highly dynamic environments such as Cloud Data Centers (DCs), mainly due to its network-wide control interface that enables fast service deployment and reconfiguration. SDN offers adequate centralisation and programmability in the routing layer that can be exploited for the development of open, fast, and infrastructure-independent NFV frameworks to facilitate cross-platform innovation through the use of open interfaces. SDN and NFV are complementary technologies and can be functional building blocks of each other: SDN can be exploited to dynamically isolate and route traffic to specific NFs by abstracting the physical topology, while NFV can create the virtual infrastructure upon which further SDN abstractions (e.g., virtual networks) can be instantiated.

In this paper, we propose GLANF (Glasgow Network Functions), a generic and open NFV ecosystem for Software-Defined Networks that tackles the above shortcomings and has the following main characteristics:

Container-based: NFs are encapsulated in light-weight Docker containers to provide fast instantiation time, platform-independence, high throughput and low resource utilisation.

The software and datasets used in this paper can be found at: http://dx.doi.org/10.5525/gla.researchdata.165

Transparent: Hosts do not need to change their traffic's destination to use NFs, as re-routing the traffic is handled entirely by the network without modifying packet headers.

Infrastructure independent: Traffic routing for NFs is handled separately from the DCs generic routing policies, allowing forwarding of traffic from any host to ephemeral NFs in OpenFlow-enabled environments.

Open innovation: The development of new NFs is not hindered by limitations of any particular NFV toolkit, framework or architecture. Sharing NFs in public or private repositories alleviates redundant implementations and promotes collaborative development, innovation and better software quality.

The main contributions of this work can be summarised as:

1) We advocate and evaluate the use of open, light-weight containers as platform for virtual NFs. We present wide-range of exemplar NFs implemented for GLANF.
2) We present a system that seamlessly and transparently routes traffic between end-hosts (physical or virtual machines) and NFs in generic OpenFlow-enabled networks.

The remainder of the paper is structured as follows: Section II discusses existing work. Section III presents the design and implementation of the proposed system. Section IV evaluates GLANF container performance in terms of throughput, latency and NF instantiation time. Finally, Section V concludes the paper.

II. Related Work

Middlebox virtualization and the development of NFV has attracted considerable research effort in recent years.

ClickOS [11] focuses on the design and implementation of a Xen-based hypervisor optimized for middlebox processing. Although it provides high processing performance, the platform relies on a specific programming environment (Click) and a modified, Xen-based hypervisor to run NFs. It would therefore require considerable effort to create new NFs and integrate ClickOS to existing infrastructures. In this paper, we present a framework for NFs using container-based virtualization, that provides a generic Linux system for NF implementations as well as the routing mechanism between the VMs and the NFs.

Cloud4NFV [12] has been recently proposed to manage NFs following the ETSI guidelines [3]. Their paper focuses on service chaining and deployment over a Cloud environment, but it gives no information on the interactions between the NFV and SDN layer or the routing mechanisms used. While they use fully virtualized VMs, GLANF exploits containers, offering a significantly more lightweight solution for NFs (*cf.* Section IV).

Container-based virtualization [13] is a scalable, high-performance alternative to hypervisors where the virtualization layer runs as an application within the operating system. In this approach, a commodity, general-purpose operating system runs on the hardware and hosts several isolated containers that share the same kernel. The main advantages of this approach is efficiency, lack of hypercalls overhead, and high container-per-host density. However there are isolation and security issues around containers, a recent Gartner report [14] shows that Docker, our container system of choice is production ready and using the supported security safeguards (such as SELinux, AppArmor, network namespaces and kernel versions), the technology is mature enough to be used in public PaaS environments. For GLANF, containers were chosen to provide high forwarding performance and low instantiation time.

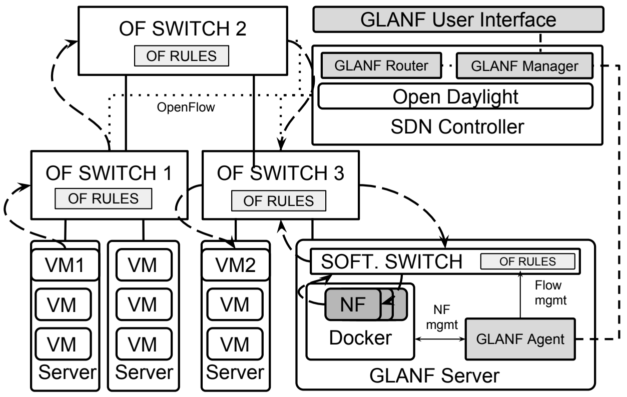


Fig. 1: High-level system architecture. Dashed arrows show the traffic flow from VM1 to VM2. Packets are routed through the associated NF hosted at the GLANF Server and are subsequently forwarded to their destination.

Docker [15] is an open platform for developers and system administrators to develop, ship and run distributed applications as containers. It has quickly become the platform of choice for container-based virtualization with recent support in Amazon Elastic Beanstalk, Microsoft Azure and Google Cloud Compute Engine. Its two core components are the *Docker Engine*, a portable, lightweight run-time and packaging tool, and the *Docker Hub*, a cloud service to share containers. Docker, on top of process (container) management, provides layered image management based on the Advanced multi-layered Unification FileSystem (AUFS) allowing incremental updates of a container image.

III. Design

A. *Traffic Routing*

Typically, middlebox policies are enforced in two ways. Either by placing the middlebox directly in the traffic path or by adding dedicated routing entries to redirect traffic to the middlebox. The first allows the middleboxes to be placed on the shortest path, however it requires infrastructural changes and result in poor flexibility as the policy will be applied to all traffic. The second allows more flexibility as an arbitrary policy chain can be configured through custom routing entries, at the cost of a longer path and overloaded routing tables, making it hard to configure and maintain [16]. In our approach, we reroute traffic to the GLANF Server hosting the relevant network function. This approach enables dynamic placement of the NFs and uses the same hosts for compute and network functions, reducing equipment costs and machine

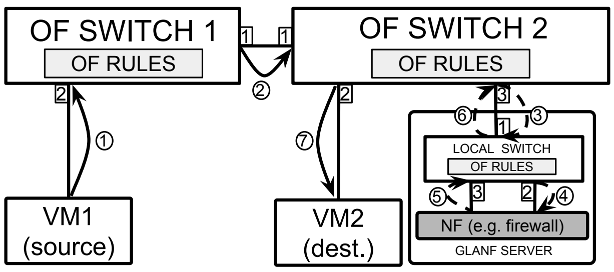


Fig. 2: Imposing a NF to selected traffic. Numbers next to the links (in boxes) denote the port number at the switch. Numbers in circles next to the arrows show the order of traffic flow from VM1 to VM2 through a NF. Table I shows the associated OpenFlow rules installed.

TABLE I: OpenFlow rules to forward packets from VM1 to VM2 through its NF using the architecture at Fig. 2.

Switch	Match	Action
1	input_port: 2, src_ip: VM1	output_port: 1
2	input_port: 1, src_ip: VM1	output_port: 3
local	input_port: 1, src_ip: VM1	output_port: 2
local	input_port: 3, src_ip: VM1	output_port: 1

specialisation. A consequence of traffic redirection is the use of non shortest path routing between source and destination possibly impacting performance, however in low-latency, high-throughput environments such as DCs, the impact is minimal as long as the number of extra hops is kept low by placing the NFs close to their associated VM [17].

GLANF relies on OpenFlow to match and forward traffic to a NFV host. Routing is handled by matching on input port, source IP and source port (depending on the routing policy used) and forwarding matched packets to an output port on the switch. As packets are never modified, routing through a NF is fully transparent to the end-hosts. By having a centralized control plane with a global view of the network, the input and output ports for the OpenFlow flow entries can be retrieved and problems of manual route (re)configuration in large-scale middlebox deployments [8] can be alleviated. In our infrastructure, *GLANF Router* is responsible for routing the traffic to the *GLANF Servers* hosting the network function. The separation of the default routing policy from GLANF's routing, achieved by using high flow priorities, allows GLANF to be deployed on any OpenFlow-enabled infrastructure without altering normal operation and changing the already installed flows.

Figure 2 shows the default traffic path using a shortest path (solid arrows), going from source VM1 to the destination VM2 through two OpenFlow-enabled switches. On the use of a network function, traffic must be redirected to the NFV server and to a particular NF (dashed arrows in Figure 2). To achieve this, the OpenFlow rules shown in Table I are inserted to the switches. Since default routing is still in place, there is no need for extra rules to route traffic at the egress of the NFV server to the destination (from OF SWITCH 2 to VM2). For clarity, only the forward path from source to destination is shown here. The reverse path from the destination back to the source is a simple inversion of the ports in Table I.

OpenFlow switches have complex trade-offs between performance and scale. The TCAM used for partial flow matching is small in size, capable of holding only a few thousand (2000–4000) flow entries [18]. It is typically collocated with a highly-specialized table holding 100,000+ entries for MAC address matching [19]. As the number of network functions increases, the growth of the flow table needs to be considered. Using only the TCAM, a single switch can redirect traffic to a maximum of 1000 NFs, as 2 entries are required per switch on the redirected path. However, only 2 flow entries are required for a collocated service chain (regardless of its length) or multiple hosts under the same CIDR mask. Finally the flow entries shown in Table I rely on L3 matching but could easily be replaced by MAC address matching using the specialized table and leaving the TCAM only for *Selective Routing* as described below.

We have identified the following types of applicable routing policies, dependent on the nature of the network function.

Exhaustive routing: All traffic from and to the host(s) goes through the NF, allowing an inspection and alteration of the entire traffic at Layer 2 or 3. Common functions requiring exhaustive routing include Intrusion Detection and Prevention (IDPS), firewalls, and Virtual Private Network (VPN) services.

Selective routing: A subset of services are routed through the NF, while the rest of the traffic follows the default route. This approach reduces the traffic load in the traversed NFs, allowing better scalability and denser network function collocation. DNS load balancers and transparent web-proxies can use this type of routing as they operate on Layer 4 with a specific service port (e.g. 53 for DNS and 80 for HTTP).

Replica routing: A replica of the traffic is routed to the NF, accounting for services that only inspect but never modify data, such as, e.g., monitoring, intrusion detection, and traffic characterisation middleboxes. Doing so, prevents performance degradation on the data path such as additional latency. Once a packet has traversed the service chain it can be discarded.

B. NFV Servers

The driving force behind NFV is to reduce cost and unify the infrastructure through low-cost commodity x86 servers. Through a common hardware base for compute and management, procurement can be more efficient, machines can be re-purposed to accommodate demand, and the maintenance cost reduced as network operators use a compatible and well-understood environment. With the significant performance improvement offered by recent software-based middlebox architectures, such as ClickOS [11] and Vyatta [20], legacy dedicated hardware appliances can be deprecated, alleviating vendor lock-in, and improving reuse and innovation through exploitation of common software practices.

To enable the instantiation and management of network functions in commodity servers, our implementation relies on the *GLANF Agent*, a single daemon running on the servers

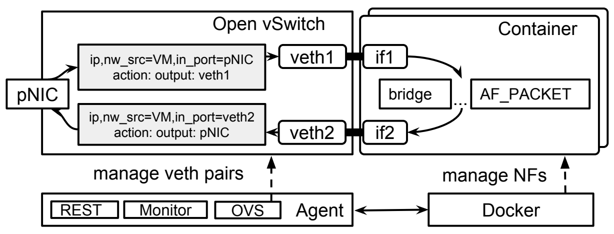


Fig. 3: Agent's network configuration for a single container.

hosting network functions. The daemon is responsible for retrieving the requested network function from the repository, instantiating and running it, routing the traffic locally to the relevant container, managing service chains, and providing information on the temporal resource and status of the host.

Our implementation relies on Docker containers for network functions that can be versioned, shared, shipped and tested with a low resource utilisation overhead. At the same time, GLANF can be used with other virtualization techniques, e.g. containers (LXC, Rocket, openVZ) or VMs (XEN, KVM or even ClickOS), as long as it provides the ability to attach two virtual interfaces (egress and ingress) provided by the Agent. However, without Docker, the system would lose some of its convenient features, such as incremental image management, public image repository and the inherent speed of deployment of docker containers.

As multiple network functions can be co-located on the same host, it is necessary to locally route traffic to each network function so that each receives the subset of traffic for which it was instantiated. Local routing of traffic is also important to respect data privacy in a multi-tenant environment and improves middlebox performance by only processing designated traffic. To route traffic within a single host, the GLANF Agent configures a local OpenvSwitch software switch by inserting and deleting OpenFlow flow entries.

Figure 3 provides an overview of the responsibilities of the GLANF Agent. It exposes a REST API to the management network, monitors the health of the machine, and communicates with Docker and OpenvSwitch to instantiate and configure the network functions.

C. Management & Orchestration

Traffic management and network function implementation are the enablers of NFV, yet for the technology to be deployed flexibility is paramount. The ability for users and operators alike to create and instantiate new network functions in minutes or hours instead of weeks and months provides unprecedented agility as well as short and cost-effective development cycles. The Management and Orchestration (MANO) framework by the European Telecommunications Standards Institute (ETSI) describes the requirements and challenges of operating a NFV infrastructure [21]. The infrastructure must be managed at different layers from the individual network services to the global network-wide requirements to maximize resource utilisation while maintaining performance guarantees.

Our implementation relies on the collaboration between the GLANF Router, Manager, Agent and UI to provide a global view and control over the infrastructure as shown in Figure 1. *GLANF Manager* is an OpenDaylight module collocated with the GLANF Router that provides a set of REST APIs to globally control the lifecycle of network functions through *create*, *start*, *stop* and *delete* primitives. It also continuously collects health status of hosts and network functions and notifications raised by the NF components. On network function instantiation, the manager selects a suitable host with the most available resources and communicates with the host's GLANF Agent to retrieve and start the requested NF from the container repository.

The decoupling the coordination logic (Manager) from the operational logic (Agent) allows for a more flexible orchestration of the infrastructure by delegating the responsibility for placement and routing to the Manager, and the network function implementation and operation to the Agent. The Manager can use different placement algorithms for the NFs in order to reduce the latency and workload of a specific machine, plan for future demand and capacity requirements based on already deployed network functions, or handle faults by transparently migrating the network function to a different server. Future work will look into placement algorithms providing different properties such as increasing NFV server utilisation or resilience, reduce latency or network hotspot prevention.

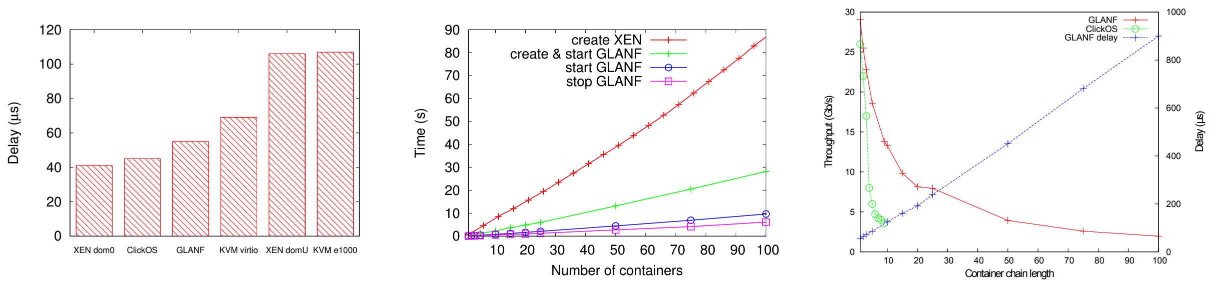
The GLANF UI is a web application that communicates with the Manager API and the northbound interface of OpenDaylight to display topological, health and status information, and to register new GLANF servers and manage network functions for one or multiple hosts. The network status is continuously updated to reflect the current state of the network and to alert users of new notifications raised by one of the running network functions.

IV. Evaluation

A. Base Performance Evaluation

In this section, we provide the base throughput, delay and boot time evaluation for multiple GLANF containers, contrasting it to other prominent function virtualization approaches. In Section IV-B, we also present exemplar implementations of useful network functions currently available over GLANF. Our experimental testbed consists of Intel i7 servers with 16GB of memory and Gigabit OpenFlow switches.

1) Throughput: We have used *iperf* to measure maximum throughput between the source and destination hosts connected via chained *wire* NFs. The wire functionality is a standard Linux bridge forwarding the traffic from the ingress to the egress ports of the NF. It is therefore the simplest form of NF and can be used to evaluate the minimum performance impact of a virtualized service. With TCP making up over 99.9% of the traffic in DCs [22], the following experiments use a single MSS-sized TCP stream and the default networking stack configuration of Linux kernel 3.13 (TCP Cubic; initial congestion window of 10 segments; minimum retransmission



(a) Idle ping delays

(b) Create/Start/Stop time

(c) Throughput and delay of chained NFs

Fig. 4: Performance evaluation of GLANF container NFs.

timeout of 200ms; window scaling, timestamps, selective acknowledgement and server-side ECN).

Figure 4c shows the packet processing throughput with NF chains hosted on a single host and is therefore not limited to the speed of the topology (e.g., 1Gbps physical switches or network cards). In this figure, we show that the container-based approach to NFs significantly outperforms ClickOS, with a single wire GLANF NF outperforming a ClickOS wire by more than 4Gb/s. It is also evident that GLANF scales better as the NF chain grows: with 9 containers chained together, packets are processed at 13.8Gb/s compared to 3.6Gb/s using ClickOS, while over Gigabit speeds can be still maintained for 100 chained containers. Note that these results are limited to the wire NFs and can be explained by the fact that GLANF does not copy the packets from the kernel space to the user space as ClickOS does.

2) Boot times: To provide high flexibility in container placement, it is necessary to quickly manage the container's lifecycle, i.e., *create*, *start* and *stop*. With a rapid turnaround time, it is possible to enable/disable new network functions in short timescales, as well as to allow for fast migration, better consolidation and placement. In case of unexpected interruption, such as container or host failure, fast recovery can be achieved by restarting the same network function on another server and redirecting traffic.

Figure 4b shows the time required to create, start and stop 1 to 100 containers on the same host. In all three cases the growth is linear, allowing the infrastructure to scale when a large number of containers are instantiated. The significant difference between *create & start* and *start* shows that it is beneficial to proactively create frequently-used containers and only starting them when required. We can observe that containers significantly outperform the creation time of Xen, and highlight the poor and exponential cost of Xen to create more domains [23].

3) Delay: Middleboxes should process packets transparently, therefore keeping additional latency to a minimum in order not to compromise end-user experience. Also, it is often required to chain multiple services to provide different network functions such as a web-cache followed by a load balancer, calling for extremely low delay for each NF in the chain.

For the results shown in Figure 4a and 4c, we have used a simple ICMP ping between source and destination to measure the idle delay impact of a wire NF. Figure 4a adds GLANF's idle latency delay impact to the original ClickOS performance evaluation [11], and compares it to ClickOS, different Xen domains, as well as to different KVM vNIC drivers. Using a stock configuration of Ubuntu Server 14.04, GLANF performs better than KVM regardless of the vNIC driver used and Xen guest system (domU). The ClickOS design aimed at providing high performance, low delay NF through significant modifications to the hypervisor resulting in a reduction of latency from 106μs (domU) to 45μs, 10μs faster than a GLANF container.

In order to keep each network function as a single functional block, it is necessary to be able to chain them to enforce multiple policies sequentially and at different layers of the topology. Figure 4c shows the maximum throughput achievable and delay induced by a chain of 1 to 100 NFs. We can see that the delay impact is linear as the number of chained containers increases. With 100 containers chained together on the same host, GLANF provides sub-millisecond delay. As the number of chained containers increases, GLANF performs 3.1× faster than Xen-based ClickOS with 5 containers and 3.8× faster with 9 containers, and allows a much higher number of containers to be chained together.

B. Network Function Implementations

We have implemented a number of exemplar network functions for GLANF, available through our Docker and GitHub repositories. These functions have been developed for demonstration purposes, to show the simplicity of creating and sharing functions.

1) Wire: The wire is a simple network function that forwards packets from its input to its output interface. It executes the *brinit* script (located in our base image) that creates a Linux bridge and adds the two interfaces to it. We have used this network function to evaluate the base system and to do baseline performance comparisons.

https://registry.hub.docker.com/repos/glanf/
https://github.com/glanf/

2) HTTP filter: A traffic filter has been implemented using Python's *Scapy* library and *Netfilter* queues. Our Python program intercepts all HTTP traffic and drops or accepts it depending on the packet's content. This filter can be used to block the access to websites containing blacklisted words such as, e.g., for parental control purposes.

3) Traffic control: Tc is used to control the Linux kernel Quality of Service (QoS) functions. In particular, we have implemented a traffic shaper using *tc* that limits the bandwidth to a threshold value. Tc also provides scheduling, policing and dropping of packets. Rate limiting can be useful to throttle background services such as backup transfers in order not to impact the response time of more bandwidth-critical applications such as VoIP.

4) Load balancer: A transparent DNS-based load-balancer that intercepts DNS query requests and replies for a specific domain, with a DNS query response created by the network function (using Python's Scapy library). For all other DNS requests, it forwards them to the intended original recipients.

5) Intrusion detection: SNORT is an open-source Intrusion Prevention and Detection System (IDPS) that performs real-time traffic analysis and packet logging. We have configured SNORT as an in-line IDPS using AF_PACKET with an alert monitor that sends notifications to the Manager and allows users to track intrusion incidents from the GLANF UI.

6) Firewall: Based on *iptables*, we have implemented a simple packet filter in our *glanf/firewall* image. The default configuration of rules forwards HTTP traffic only.

Acknowledgments

This work was supported in part by the UK Engineering and Physical Sciences Research Council (EPSRC) projects EP/L026015/1 and EP/L005255/1.

V. Conclusion

Network Function Virtualization offers a cost-effective alternative to purpose-built middleboxes that provide limited functionality, and are cumbersome to deploy and maintain to accommodate for emerging service requirements. However, existing implementations of NFV are significantly constrained by the static routing configuration of the underlying network infrastructure, leading to topology and operator-specific deployments.

In this paper, we have exploited SDN and container-based virtualization to devise GLANF, an open, infrastructure-independent NFV framework that can be deployed in highly-dynamic environments, and can foster innovation in the development of network functions that are missing from today's solutions. We have evaluated GLANF over a cloud testbed using an OpenFlow-enabled network and in addition to the flexibility of creating and attaching NFs to arbitrary hosts, we show significant improvement in throughput and function instantiation time by using containers over existing, hypervisor-based NFV platforms.

References

[1] J. Sherry, S. Hasan, C. Scott, A. Krishnamurthy, S. Ratnasamy, and V. Sekar, "Making middleboxes someone else's problem: network processing as a cloud service," *ACM SIGCOMM Computer Communication Review*, vol. 42, no. 4, pp. 13–24, 2012.

[2] World enterprise network and data security markets. [Online]. Available: https://www.abiresearch.com/market-research/product/1006059-world-enterprise-network-and-data-security/

[3] E. T. S. Institute. (2012) Network Functions Virtualisation, White Paper. [Online]. Available: http://portal.etsi.org/NFV/NFV_White_Paper.pdf

[4] J. Carapinha, P. Feil, P. Weissmann, S. E. Thorsteinsson, Ç. Etemoğlu, Ó. Ingórsson, S. Çiftçi, and M. Melo, "Network Virtualization - Opportunities and Challenges for Operators," in *Future Internet - FIS 2010*. Springer Berlin Heidelberg, 2010, pp. 138–147.

[5] D. King and C. Ford, "A critical survey of Network Functions Virtualization (NFV)," in *iPOP: IP Over Optical*, 2013.

[6] L. Bondan, C. Dos Santos, and L. Zambenedetti Granville, "Management requirements for clickos-based network function virtualization," in *Network and Service Management (CNSM), 2014 10th International Conference on*, Nov 2014, pp. 447–450.

[7] D. T. A. Nicolai Leymann, "Flexible service chaining. requirements and architectures." in *EWSDN: European Workshop on Software Defined Networks*, 2013.

[8] J. Sherry and S. Ratnasamy, "A survey of enterprise middlebox deployments," EECS Department, University of California, Berkeley, Tech. Rep., Feb 2012.

[9] Software-Defined Networking: The new norm for networks (ONF White Paper). [Online]. Available: https://www.opennetworking.org/images/stories/downloads/sdn-resources/white-papers/wp-sdn-newnorm.pdf

[10] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.

[11] J. Martins, M. Ahmed, C. Raiciu, V. Olteanu, M. Honda, R. Bifulco, and F. Huici, "Clickos and the art of network function virtualization," in *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14). Seattle, WA: USENIX Association*, 2014, pp. 459–473.

[12] J. Soares, M. Dias, J. Carapinha, B. Parreira, and S. Sargento, "Cloud4NFV: A Platform for Virtual Network Functions," in *Cloud Networking (CloudNet), 2014 IEEE 3nd International Conference on*, 2014.

[13] S. Soltesz, H. Pötzl, M. E. Fiuczynski, A. Bavier, and L. Peterson, "Container-based operating system virtualization: a scalable, high-performance alternative to hypervisors," in *ACM SIGOPS Operating Systems Review*, vol. 41, no. 3. ACM, 2007, pp. 275–287.

[14] Gartner. (2015, Jan.) Security properties of containers managed by docker. [Online]. Available: https://www.gartner.com/doc/2956826/

[15] Docker, the Linux Container Engine. [Online]. Available: http://www.docker.io

[16] D. Joseph, A. Tavakoli, and I. Stoica, "A Policy-aware Switching Layer for Data Centers." ACM, 08/2008 2008, pp. 51–62.

[17] F. P. Tso and D. P. Pezaros, "Baatdaat: Measurement-based flow scheduling for cloud data centers," in *2013 IEEE Symposium on Computers and Communications, ISCC 2013, Split, Croatia, 7-10 July, 2013*, 2013, pp. 765–770.

[18] B. Stephens, A. Cox, W. Felter, C. Dixon, and J. Carter, "PAST: Scalable Ethernet for Data Centers," ser. CoNEXT '12. New York, NY, USA: ACM, 2012, pp. 49–60.

[19] A. X. Liu, C. R. Meiners, and E. Torng, "TCAM Razor: A Systematic Approach Towards Minimizing Packet Classifiers in TCAMs," *IEEE/ACM Trans. Netw.*, vol. 18, no. 2, pp. 490–500, Apr. 2010.

[20] Vyatta routing platform. [Online]. Available: http://vyatta.org

[21] E. T. S. Institute. (2013) Network Functions Virtualisation (NFV); Architectural Framework, White Paper. [Online]. Available: http://www.etsi.org/deliver/etsi_gs/NFV/001_099/002/01.01.01_60/gs_NFV002v010101p.pdf

[22] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. a. Prabhakar, S. Sengupta, and M. Sridharan, "Data Center TCP (DCTCP)," ser. SIGCOMM '10. New York, NY, USA: ACM, 2010, pp. 63–74.

[23] P. Harvey and J. Sventek, "Wireless sensor network simulation with Xen," in *Proceedings of the 46th Annual Simulation Symposium*. Society for Computer Simulation International, 2013, p. 4.