GNFC: Towards Network Function Cloudification

Richard Cziva, Simon Jouet and Dimitrios P. Pezaros
School of Computing Science, University of Glasgow, Glasgow, G12 8QQ, Scotland
{richard.cziva, simon.jouet, dimitrios.pezaros}@glasgow.ac.uk

***Abstract*—An increasing demand is seen from enterprises to host and dynamically manage middlebox services in public clouds in order to leverage the same benefits that network functions provide in traditional, in-house deployments. However, today's public clouds provide only a limited view and programmability for tenants that challenges flexible deployment of transparent, software-defined network functions. Moreover, current virtual network functions can't take full advantage of a virtualized cloud environment, limiting scalability and fault tolerance.**

In this paper we review and evaluate the current infrastructural limitations imposed by public cloud providers and present the design and implementation of GNFC, a cloud-based Network Function Virtualization (NFV) framework that gives tenants the ability to transparently attach stateless, container-based network functions to their services hosted in public clouds. We evaluate the proposed system over three public cloud providers (Amazon EC2, Microsoft Azure and Google Compute Engine) and show the effects on end-to-end latency and throughput using various instance types for NFV hosts.

I. Introduction

Enterprise networks rely on hardware-based network appliances or *middleboxes*. Middleboxes became fundamental parts of networks, providing approx. 45% of the network devices to enforce security (e.g., firewalls and intrusion detection systems), performance (e.g., rate limiters, proxies, load-balancers) and reduced bandwidth costs (e.g. WAN optimizers) [24]. Studies also show that demand for in-network processing will increase even further with the advent of diverse consumer devices and multimedia protocols [2]. However, traditional, hardware-based middleboxes have many 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 be typically extended to accommodate new functionality as new operational requirements emerge. The proprietary software and hardware limit innovation and create vendor lock-in [18].

Network Function Virtualization (NFV) is a novel approach to address the above shortcomings of managing closed and proprietary appliances through 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) [9]. By using Software-Defined Networking (SDN) to manage NFV, the compatibility with existing deployments can be simplified, and operation and maintenance procedures can be facilitated. NFV has already gained a considerable momentum: seven of the world's leading telecom operators along with other 52 network operators, IT vendors and technology providers have initiated a new ETSI standards group [17].

With the rise of public cloud computing, enterprises have started to migrate resources to the cloud in order to increase agility and scalability and to reduce maintenance costs of their infrastructure. According to Forrester, adoption of public clouds by enterprises will cross over the 36% mark this year [23]. Despite the growing adoption, key challenges remain when migrating enterprise network services to the cloud - including performance, privacy, and security issues [16]. Underlying many of these challenges is that network configurability provided by public clouds is still very limited (or absent) compared to enterprise's in-house counterparts. The cloud networking model has largely focused on providing basic reachability using private and public IP addresses attached to the VMs and simple NFs such as firewalls or basic load-balancers. While NFs are crucial for enterprises, key network functions are still missing from today's public clouds, e.g. fine-grained network isolation, filtering and service differentiation, policy-based routing, protocol acceleration, control over addressing or caching for improved performance or availability [25] [10].

However, to take full advantage of a virtualized cloud environment, it is not sufficient to simply port an existing network application to run on a Virtual Machine (VM) in the cloud. To reach scalability, on-demand provisioning and fault recovery, traffic between hosts should be managed dynamically and NFs must have clear separation between their stateless application code and data stored in logically centralized cloud storage. This transformation from NFV to cloud-based virtual NFs has recently defined the term Network Function Cloudification [4].

In this paper, we present and evaluate the existing limitations of public Infrastructure-as-a-Service clouds that challenge the design of a cloud-based NFV framework. Following those limitations, we propose GNFC (Glasgow Network Function for the Cloud) an NFV framework for public clouds that allows transparent and stateless container-based NFs to be attached to arbitrary services. We present the design, implementation and evaluation of our proposed system that gives tenants the ability to create and deploy their own NFs such as network isolation, intrusion detection, service differentiation or caching and attach them to any of their VMs. Unlike traditional NFs (e.g. Sophos, Vyatta, ClickOS), GNFC NFs are stateless and transparent to the services they are attached to and can be enabled, disabled or migrated on-demand. This novel model increases flexibility and functionality, saves cost and deployment complexity compared to current approaches where traffic steering needs to be configured manually for every service using NFs.

The contributions of this work can be summarized as:

1) the review and evaluation of the limitations hindering the deployment and management of NFV in public clouds
2) the design and implementation of GNFC, a framework that allows NFs to be hosted and transparently attached to arbitrary services in the cloud
3) the experimental evaluation of GNFC over popular public clouds, comparing various instance types

II. Limitations of Public Clouds

In this Section we describe the challenges and limitations of implementing and managing NFV in public clouds.

A. Programmability

Network functions require traffic to be steered through them. In traditional networks, it is done either by manually configuring the forwarding tables of the switches and routers along the path or automatically by using an SDN controller. In public clouds, even when the network is configurable, the programmability and flexibility of the virtual network is extremely limited. Typically, the network is abstracted into a single virtual switch connecting all the VM instances together and to the Internet, and simple destination-based routing entries can be inserted. Without the ability to generate more complex virtual topologies where NFs can be placed on the traffic path or allowing finer-grained control of the forwarding policies with control protocols such as OpenFlow [21], the deployment of network functions is hindered.

B. IP Forwarding

Cloud providers such as Amazon and Google have IP forwarding functionality disabled by default in their clouds and it can only be enabled when a VM is created. This inability to change the IP forwarding policy of instances can hinder deployment of transparent NFs in existing infrastructures as every instance would have to be recreated to enable IP forwarding. Once IP forwarding is enabled, traffic can be steered between instances as long as the source and destination of the packet is local to the provider's private network and is confined to the tenant's set of virtual machines. As middleboxes typically apply policies on ingress Internet traffic (e.g., IDPS, Firewall, VPN, WAN accelerator), limiting IP forwarding to only private traffic prevents the use of a VM instance as an Internet Gateway and therefore as a simple mechanism for middlebox deployments.

C. Protocol support

In order to simplify Network Address Translation (NAT) and shape the traffic to improve Quality of Service (QoS), most public cloud providers only allow traditional transport protocols (e.g., TCP, UDP, ICMP) to be used by the VMs. Layer 2.5 protocols such as MPLS and Layer 3 protocols such as IPsec, IP-in-IP or GRE are commonly not supported, preventing traditional traffic encryption and encapsulations mechanisms to be used.

TABLE I: Support for GRE and VXLAN tunneling protocols in public cloud providers.

	EC2	**Azure**	**GCE**
GRE	✓	✗	✗
VXLAN	✓	✓	✓

D. Support for NFV frameworks

Today's popular NFV frameworks (such as the high-performance ClickOS [20] or Brocade Vyatta [1]) are often built on top of specific software components or require network configuration that is rarely supported by most public providers. ClickOS for instance uses a highly modified XEN hypervisor to improve network performance that prevents its deployment on public clouds. Vyatta requires cloud-specific network configuration and therefore cannot be ubiquitously deployed over public cloud infrastructures.

III. GNFC Architecture

To address the described limitations in Section II, we designed GNFC, a framework for deployment and management of NFs in public cloud environments. GNFC provides components for centralized management of NFs (Section III-A), transparent and flexible traffic steering using common tunnelling technologies and OpenFlow (Section III-B), instantiation of container-based middleboxes (Section III-D), and a user interface for the system (Section III-E). Through a set of well defined APIs exposed by each component, GNFC gives programmability to rigid networking environments, allowing Layer 2 packets to be selectively and transparently steered through one or more NF.

A. Orchestration

The system is orchestrated by the GNFC Manager, a component implemented for the OpenDaylight SDN controller (as shown in Figure 1). The Manager maintains a connection with the registered Agents that start and stop NFs and send notifications and status information to the Manager. It supervises the running NFs and the available GNFC servers (VMs running the GNFC Agent), and allocates new NFs to the least loaded GNFC server. It also stores the incoming notifications from the NFs and exposes the main REST API to orchestrate the entire system.

B. Traffic Management

To address the limitations of traffic forwarding while achieving transparent traffic steering in public cloud networks (described in Section II), GNFC relies on tunnelling protocols such as Virtual Extensible LAN [19] (VXLAN) or Generic Routing Encapsulation [11] (GRE). By using tunneling protocols, arbitrary Layer 2 (Ethernet) traffic can be encapsulated within an IP or UDP frame and forwarded to any other host using traditional routing mechanisms without modification of the tunneled traffic. Other encapsulations, such as IP-in-IP can be used, however as it encapsulates IP packets, it would prevent the middleboxes from processing non-IP traffic such as L2 broadcast messages or IPv6. Another option can be

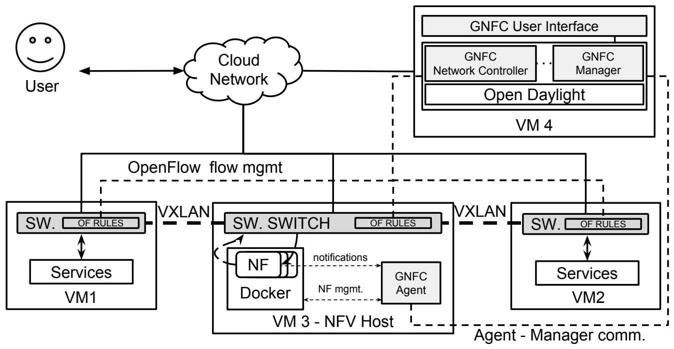

Fig. 1: High-level overview of GNFC.

using TCP tunnels between all VMs (e.g. an SSH tunnel), however, using a reliable transport layer would deteriorate encapsulated real-time application traffic bound by response time instead of reliability (such as VoIP, media streams). Therefore by encapsulating Ethernet frames in IP (GRE) or UDP (VXLAN) packets, flexible and transparent traffic steering can be achieved without relying on programmable devices within the network.

Once the forwarding limitation has been addressed through tunneling, the problem of protocol support arises. As VXLAN encapsulates Ethernet frames within UDP datagrams while GRE uses IP packets, it would be sensible to use GRE for a lower encapsulation overhead, leading to higher MTU and therefore higher goodput. However, as shown in Table I and discussed in the Section II, providers such as Azure and Google GCE drop any traffic that is not TCP, UDP or ICMP, preventing GRE to be used by tenants. With the universal support of UDP, VXLAN tunnels can be used in today's public cloud providers to configure tunnels at the source and destinations hosts to steer traffic through a third VM acting as the middlebox. However, to provide the ability to quickly, elastically start and stop NFs, programmability and centralisation are necessary.

Programmability can be achieved by using programmable software switches on the VMs. Controlling them using the SDN paradigm with a management API such as OpenFlow allows the network to be configured on-demand and provides the flexibility required to centrally and quickly redirect traffic through dynamically instantiated NFs. Open vSwitch (OvS), a popular and high-performance software switch has been used to create and manage tunneled traffic between the VMs in the cloud. As the recent versions of OvS support flow-based tunneling for both GRE and VXLAN protocols, allowing the configuration of the tunnels within the flow table. By using OpenFlow's Write-Metadata instruction, the tunnel's unique id (VXLAN Network Identifier or GRE Key) as well as the remote host (the host running the NF) is specified by every individual flow entry, allowing the controller to manage every tunnel centrally.

TABLE II: OpenFlow rules required at the VMs to forward packets between VM1 to VM2 through NF1 hosted on VM3.

Hop	VM	Match	Action

1	VM1	in_port=LOCAL
		nw_dst=VM2
			set_tunnel:100
			set_field:VM3->tun_dst
			output:tunnel

2	VM3	in_port=eth0
		tun_id=100
			output:veth1 (NF1's input)

Middlebox processing of VM1 → VM2 packets in NF1			
3	VM3	in_port=veth2 (NF1's output)	set_tunnel:101 set_field:VM2->tun_dst output:tunnel

4	VM2	in_port=LOCAL
		nw_dst=VM1
			set_tunnel:102
			set_field:VM3->tun_dst
			output:tunnel

5	VM3	in_port=eth0
		tun_id=102
			output:veth1 (NF1's input)

Middlebox processing of VM2 → VM1 packets in NF1			
6	VM3	in_port=veth2 (NF1's output)	set_tunnel:103 set_field:VM1->tun_dst output:tunnel

7	VM1	in_port=eth0
		tun_id=103
			output:LOCAL

C. Network Controller

To centrally control the configuration of the tunnels and insert the forwarding rules necessary to steer the traffic to the appropriate NFs, we have designed the *GNFC Network Controller*. This network controller is an SDN application built on top of the OpenDaylight controller framework and collocated with the GNFC Manager. To support different types of NFs, GNFC implements three types of forwarding policies:

1) Exhaustive forwarding: All traffic coming from or going into a VM goes through a NF, allowing inspection and alteration of the entire traffic. Popular middleboxes require exhaustive forwarding Intrustion and Prevention Systems (IDPS), firewalls or VPN services.
2) Selective forwarding: A subset of services (selected by ports) are routed through a NF, while other services use the default forwarding policies. This type of forwarding allows efficient redirection for load balancers or proxies as they usually only require the traffic associated to a specific service port (e.g. UDP 53 and TCP 80).
3) Replica forwarding: This policy has been added to support passive inspection of the traffic without modifying it. This policy can be used for monitoring, intrusion detection (IDS) or traffic characterisation middleboxes.

Figure 1 shows an example deployment of GNFC, where VM1 and VM2 are hosts running tenant specific workload, while VM3 (NFV host) runs the GNFC Agent and is therefore capable of hosting NFs. VM4 hosts the network controller and the manager to centrally orchestrate the system. In Table II, we show the OpenFlow flow entries installed at the software switches to traverse all traffic between VM1 to VM2 through a NF hosted on VM3 using the exhaustive forwarding policy. The next hop of the packets is set with *set_tunnel* and *set_field* actions before the packets are forwarded to the local tunnel interface (hop 1, 3, 4, 6 in Table II). The flows in Table II

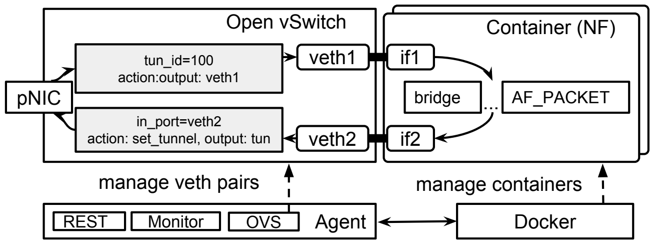

Fig. 2: GNFC Agent.

also ensure that the replies from VM2 to VM1 are forwarded through the same NF.

D. Network Functions

While different, software-based middlebox platforms and solutions such as ClickOS [20], FlowOS [8] and Vyatta [1] exist, GNFC uses unmodified Docker [22] containers to encapsulate stateless middlebox functionality. Docker provides a lightweight equivalent to VMs, allowing a much higher NF-to-host density and smaller footprint at the cost of reduced control over the kernel. On top of process and network isolation, Docker delivers a way to version, reuse and store container images in public or private repositories. Docker is available on any Linux-based distribution, enabling its use over every public cloud infrastructure, overcoming the limited support of current NFV platforms described in Section II.

GNFC NFs are managed by the GNFC Agent that installs and talks to the local Docker daemon to retrieve, start, stop and delete containers. The Agent provides a REST API for the Manager and sends status information and notifications from the NFs to the Manager. On a create request from the Manager, the Agent calls the local Docker daemon to pull the images from the Docker public or private repository if the image is not already available locally. Then, the Agent creates a new container based on this image, registers it locally as a GNFC managed container and sends an event on the event stream to the Manager. While the containers are started, the Agent creates two virtual Ethernet pairs (veth pairs) for the ingress and egress and injects one end of the pair inside the container's namespace and the other end to the local software switch, as shown in Figure 2.

GNFC NFs are able to send notifications (arbitrary messages) to the users by calling a REST endpoint accessible by every container, provided by the local Agent. When the Agent receives a notification from an NF, it forwards them to the Manager to centrally store every NF notification.

E. User Interface

A web-based User Interface for GNFC has been designed and implemented. The UI provides a user-friendly and visual way to deploy, manage and monitor NFs and attach them to services running on VMs in their cloud. It also allows the inspection of the notifications sent by the NFs to monitor the health and status of the system.

IV. Evaluation

In this Section, we show the evaluation of GNFC over three popular public clouds, namely Amazon EC2, Google GCE and Microsoft Azure. The evaluation focuses on the overhead (throughput and latency penalty) of the proposed traffic management using traffic tunneling methods.

All measurements were taken in parallel, within a short time period to avoid differences in network characteristics. The VMs were started in the Western European region of the providers. The measurements have been repeated 10 times at every measurement round. Each measurement round involves the request of new instances from the cloud provider to get a new and randomly allocated set of participating VMs. When VMs got co-located in a measurement round (giving a pairwise throughput of 4-5 Gbit/s), our measurement script requested new VMs to exclude this scenario. High-performance instances (c4.large in EC2, D2 in Azure, n1-standard-2 in GCE) were used as the source and destination VMs to eliminate bottlenecks. Regardless of the provider, each VM used the default image of Ubuntu Server 14.04 LTS provided with an unmodified Open vSwitch 2.0.2. To measure the throughput both in UDP and TCP mode we have used the Iperf utility.

Five experiments have been conducted for this paper. First, the tunneling overhead through various instance types is shown comparing the three providers (Section IV-A and IV-B). In Section IV-C, GRE and VXLAN tunnels have been compared over EC2. Finally, Section IV-D evaluates the overhead of GNFC NFs compared to their static deployments.

A. Throughput

In this Section, we show the end-to-end throughput of the traffic steered through a middlebox using VXLAN tunnels. To evaluate the impact of the different instance types as middleboxes on the network performances each instance is simply running a software switch NF, forwarding transparently the traffic from the source to the destination and vice-versa. Figure 3 compares the UDP and TCP performance of the direct traffic from source to destination (baseline) to the maximum throughput achievable when the traffic is steered through a software switch NF (a software switch NF forwards all packets without modification). Most cloud's network performance is directly related to the instance type requested by the user, with overall better network performance in terms of throughput, latency and stability for more expensive instances. To evaluate these differences, we used instance types ranging from a price per hour of $2.450 (Azure A8) to $0.012 (GCE f1-micro). Therefore, the instance name shown in the x-axis of the figure represents the instance type of the VM hosting the NF, for instance, t2.micro represents the throughput from a c4.large instance to a c4.large instance through a software switch NF hosted on a t2.micro instance.

As shown in the figure, baseline UDP performance is always lower than baseline TCP performance at all providers. This is important, as VXLAN operates over UDP and therefore any traffic that is encapsulated with VXLAN will be capped at UDP's baseline performance. However, it is not surprising

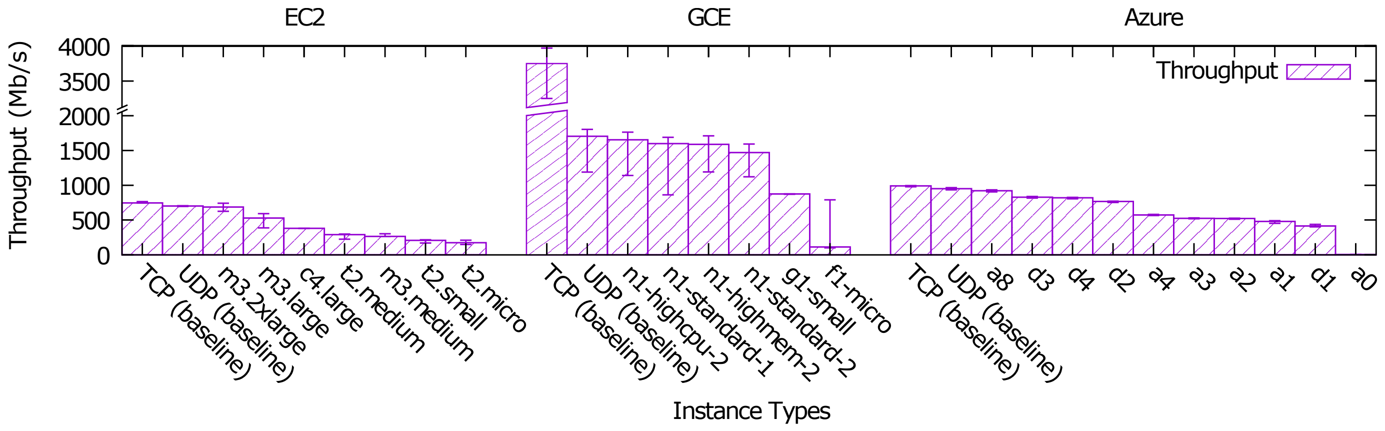

Fig. 3: RTT and end-to-end throughput evaluation on steered traffic through varying EC2, GCE and Azure VM instance types

that cloud providers apply different QoS policies between TCP and the rest of the traffic, as TCP is the dominant protocol with 99.91% of the traffic [6] and other protocols without congestion control can lead to flow unfairness, congestion and bufferbloat [15].

In general, using small and micro instances (t2.micro, f1-micro, A0) as NFV hosts, the end-to-end throughput degrades drastically. This can be attributed to the CPU shares allocated to the VMs, especially impacting smaller instances as described by Guohui Wang and Eugene Ng [26]. By using instances with more and higher frequency vCPU cores (e.g. m3.2xlarge on EC2, n1-highcpu-2 on GCE or A8 on Azure), the achievable throughput is close to the UDP baseline (maximum reachable) at all providers. It is also important to mention that the relationship between the size (price) of the instances and the achievable throughput using them as NFV hosts is not always linear. For example, in our experiments over Azure, using a compute-optimized, more expensive D1 instance we reached lower throughput than using a cheaper A1 instance.

Stability of the throughput can also be observed from Figure 3, as the error bars show the first and third quartile of the throughput. While GCE delivers the highest performance, the throughput is highly variable with a throughput 46% lower than the median for the first quartile and 6% higher for the third quartile (n1-standard-1). EC2 and Azure deliver lower but consistently stable throughput over different measurement rounds and instance types with a maximum of 27% (m3.large) and 3% (D1) differences from the median respectively.

B. Delay

As the traffic is steered through an NF hosted on a VM with an unknown physical location and distance from other VMs, it is also necessary to evaluate the impact on the average delay that such steering involves. To do so, the Round-Trip-Time (RTT) has been measured with the various instance types. The measurements were conducted using the ping utility, sending 300 ICMP requests at each measurement round, with each measurement round involving the same steps as defined in the previous Section. A large number of measurement rounds have been conducted to avoid any particular virtual machine placement. Figure 4 presents a scatter plot of the mean throughput and RTT for the various instance types we evaluated.

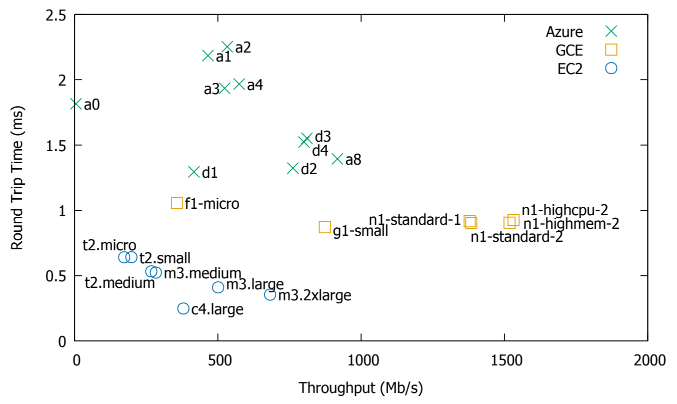

Fig. 4: Mean throughput and RTT of various instance types over Azure, GCE and EC2.

As shown in Figure 4, the RTT through EC2's c4.large instance is approx 0.3 ms, while using Azure's A2 instance, the RTT is 2.3 ms, almost 8 times higher. It can be clearly seen that Amazon's network provides the lowest latency for all instances, followed by Google's 1ms average RTT. On Azure, a high RTT has been measured, even for the large, network optimized instances (e.g., A8). We can also observe that the delay is usually higher for the cheaper (small and micro) instances at all providers. It is also interesting to observe that the RTT for Azure instances depends highly on their type, while EC2 and especially Google provide a more stable RTT across various instance types.

On top of delay properties, Figure 4 highlights three distinct clusters of instances that match the three providers. According to this scatter plot, Azure delivers high RTT and average throughput, EC2 provides low RTT and average throughput, while GCE gives high throughput and mid-range RTT. This difference in latency could simply imply that the network in

TABLE III: Comparing RTT and throughput using GRE and VXLAN tunnels over EC2.

	RTT (ms)			**Throughput (Mbit/s)**		
Instance	**GRE**	**VXLAN**	**G/V**	**GRE**	**VXLAN**	**G/V**
t2.micro	0.61	0.65	0.94	214.50	215.50	1.00
t2.small	0.64	0.62	1.03	232.50	236.50	0.98
t2.medium	0.49	0.51	0.97	292.50	300.00	0.97
m3.medium	0.48	0.47	1.01	290.50	286.50	1.01
m3.large	0.39	0.42	0.93	608.00	579.00	1.05
m3.2xlarge	0.33	0.37	0.89	745.00	696.50	1.07
c4.large	0.23	0.25	0.94	380.00	380.00	1.00
		Average:	**0.95**		Average:	**1.01**

some providers is slower due to a slower switching fabric, larger buffers or higher oversubscription. However, it could also show the performance of the placement algorithm used by the different providers, with Amazon EC2 bringing the VMs closer together than GCE or Azure and consequently reducing latency.

C. GRE vs VXLAN

As mentioned in Section III-B, GNFC can utilize two popular tunneling techniques, namely GRE and VXLAN tunnels. Due to the limited support of the GRE protocol, the measurements presented in Section IV-A and IV-B used VXLAN tunnels. In this Section, we compare the performance of both tunneling techniques over Amazon EC2, as it supports both. Table III presents the mean RTT and throughput using GRE and VXLAN tunnels over different instance types.

As shown in Table III, we couldn't measure significant difference in throughput and RTT between GRE and VXLAN tunnels over Amazon EC2. However, a slight improvement in RTT (5% on average) and increase in throughput (1% on average) can be noticed with GRE tunnels. This slight improvement might be explained with the lower encapsulation overhead of GRE, as mentioned in Section III-B.

D. GNFC NF Overhead

In this experiment, we measure the performance penalty of GNFC NFs. We compare a static deployment of four common NFs with deployments in GNFC container-based NFs. The measurement were carried out on EC2 using c4.large instances. Figure 5 presents the normalized throughput at the 90th percentile of the following example NFs[1]:

1) rate limiter: using the *tc* utility, this NF limits the traffic to a specified bandwidth
2) software switch: using a simple Linux bridge, this NF forwards all packets without modification
3) firewall: this NF relies on the well-known *iptables* to filter packets
4) snort: the classic SNORT intrusion detection software

The results have been normalized to show GNFC's impact on network performance, instead of presenting configuration and environment specific throughput, as for example SNORT's

[1]More example NFs can be found at:
https://registry.hub.docker.com/repos/glanf/

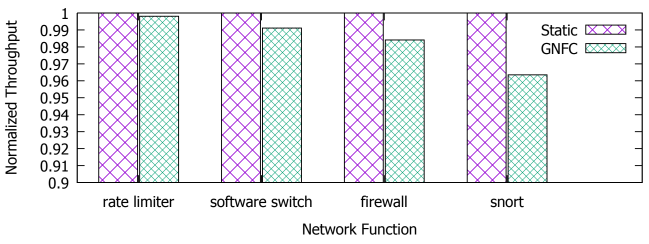

Fig. 5: Normalized throughput of few NFs using static and GNFC deployments at the 90th percentile.

throughput is highly dependent on the configured level of traffic inspection. As shown in Figure 5, GNFC's containerization adds only a minimal penalty to the throughput. The performance difference for the software switch and the rate limiter NFs is less than 1% and in the worst case for the snort NF it is maximum 3.4%. In the current implementation GNFC uses a virtual Ethernet pairs to link OvS and the NF, however multiple recent benchmarks highlight that OvS internal ports provide better performance, potentially closing the small gap between static and dynamic deployment performances [5]. For this minimal performance degradation, container-based NFs can be deployed, started and stopped by few calls on any Linux host at all public cloud providers.

V. Discussion

Network Function Cloudification or making today's network functions 'cloud-native' is a new paradigm driven by industry leaders such as Mellanox, Metaswitch Networks and HP to support agility and elasticity [4]. The re-architecting of today's NFV frameworks to be cloud-native with smaller failure domains and stateless transactions is an ongoing and challenging process that is likely to trend in the NFV communities in the near future [14]. This provides two key challenges for discussion: decoupling the state and achieving performance in the face of decoupled state.

While it is known that, e.g., Deutsche Telekom is experimenting with Docker-based NFs [3], to the best of our knowledge we are the first to publish an extensive evaluation of container-based NFs over public clouds. However the performance and security isolation provided by containers is not yet on the same level as isolation provided by traditional VMs, the technology evolved significantly during the last few years and as highlighted by a recent Gartner report, it is now mature enough for production deployments [12]. We see great potential in container NFs and in particular, in Docker. Containers require less resources (approx. 5MB memory for a running container), can scale easily and be deployed in short timescales, and allow fast prototyping and testing on commodity devices. Although we see Docker and more globally container-based virtualization as a great fit for NFV, containers can be easily replaced in GNFC with other software-based middlebox by modifying the GNFC Agent.

VI. Related Work

CloudNaaS [7] enables deployment of network functions such as network isolation or service differentiation, and allows flexible interposition of various middleboxes in a private cloud. CloudNaaS relies on a modified cloud controller and it has been designed on top of a controllable OpenFlow-enabled network. In public cloud environments, the cloud controller can't be modified and the network is fully virtualized, providing no access to the individual devices or topology. In contrast with CloudNaaS, GNFC can be used in pubic clouds, however it has no control over the network to provide optimal physical placement for middleboxes.

Sherry *et. al.* designed and implemented APLOMB [25], a service for outsourcing enterprise middlebox processing to the cloud. APLOMB outsources the vast majority of middleboxes from a typical enterprise network with impact on performance, while providing a scalable, affordable solution for middlebox processing. In GNFC, we present a way to use middlebox processing for VMs already deployed in a public cloud. By hosting both VMs and NFs in the cloud, the additional latency and data transfer cost introduced by APLOMB can be eliminated.

Authors of Stratos [13] argue that today's cloud middlebox deployments lack the same abstraction that are used for compute or storage services in the cloud. Stratos allows tenants to specify logical middlebox deployments and provides efficient scaling, placement and distribution algorithms that abstract away low-level issues ensuring effective application performance. While Stratos can operate in public clouds delivering efficient scaling and chaining of middleboxes, GNFC focuses on the deployment of virtual, container-based middleboxes that can be transparently attached to VMs in the cloud.

VII. Conclusion

Providing transparent and flexible middlebox processing for applications in today's public clouds is challenging due to the limited programmability and access to the networks offered by public clouds and the design of current NFV platforms. As an approach to overcome these challenges, this paper presents GNFC, a container-based NFV platform for public clouds. GNFC enables end-users to attach stateless network functions to their VMs and services by utilizing tunneling techniques and software-defined networking. GNFC has been evaluated over three popular public cloud providers (Amazon EC2, Google Compute Engine, Microsoft Azure) to present how various instance types impact the end-to-end throughput and RTT of the proposed traffic management. We show significant differences between various cloud providers that needs to be taken into account by tenants when planning to move their network functions to the cloud.

Acknowledgments

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

References

[1] Vyatta routing platform. http://vyatta.org.

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

[3] Deutsche Telekom experimenting with NFV in Docker. http://www.businesscloudnews.com/2015/02/09/deutsche-telekom-experimenting-with-nfv-in-docker/, February 2015.

[4] From Network Function Virtualization to Network Function Cloudification: Secrets to VNF Elasticity. http://www.mellanox.com/blog/2015/03/, March 2015.

[5] Open vSwitch Internal Ports and Linux veth Devices . http://flavioleitner.blogspot.co.uk/2015/05/, May 2015.

[6] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. a. Prabhakar, S. Sengupta, and M. Sridharan. Data Center TCP (DCTCP). In *Proceedings of the ACM SIGCOMM 2010 Conference*, SIGCOMM '10, pages 63–74, New York, NY, USA, 2010. ACM.

[7] T. Benson, A. Akella, A. Shaikh, and S. Sahu. CloudNaaS: a cloud networking platform for enterprise applications. In *ACM SOCC*, page 8, 2011.

[8] M. Bezahaf, A. Alim, and L. Mathy. FlowOS: A Flow-based Platform for Middleboxes. In *HotMiddlebox '13*.

[9] 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 *FIS 2010*. 2010.

[10] Cloudorado. Cloud computing comparison engine. https://www.cloudorado.com, 2015.

[11] D. Farinacci, P. Traina, S. Hanks, and T. Li. Generic routing encapsulation over ipv4 networks. 1994.

[12] Gartner. Security properties of containers managed by docker. https://www.gartner.com/doc/2956826/, Jan. 2015.

[13] A. Gember, A. Krishnamurthy, S. S. John, R. Grandl, X. Gao, A. Anand, T. Benson, V. Sekar, and A. Akella. Stratos: A Network-Aware Orchestration Layer for Virtual Middleboxes in Clouds. 2013.

[14] A. Gember-Jacobson, R. Viswanathan, C. Prakash, R. Grandl, J. Khalid, S. Das, and A. Akella. OpenNF: Enabling innovation in network function control. In *Proceedings of the 2014 ACM conference on SIGCOMM*, pages 163–174. ACM, 2014.

[15] J. Gettys and K. Nichols. Bufferbloat: Dark buffers in the internet. *Queue*, 9(11), Nov. 2011.

[16] M. Hajjat, X. Sun, Y.-W. E. Sung, D. Maltz, S. Rao, K. Sripanidkulchai, and M. Tawarmalani. Cloudward Bound: Planning for Beneficial Migration of Enterprise Applications to the Cloud. *SIGCOMM 2010*, 40(4):243–254, Aug. 2010.

[17] E. T. S. Institute. Leading operators create ETSI standards group for network functions virtualization.

[18] E. T. S. Institute. Network Functions Virtualisation, White Paper.

[19] M. Mahalingam, D. Dutt, K. Duda, P. Agarwal, L. Kreeger, T. Sridhar, M. Bursell, and C. Wright. Virtual extensible local area network (VXLAN): A framework for overlaying virtualized layer 2 networks over layer 3 networks. *Internet Req. Comments*, 2014.

[20] J. Martins, M. Ahmed, C. Raiciu, V. Olteanu, M. Honda, R. Bifulco, and F. Huici. Clickos and the art of network function virtualization. In *NSDI 2014*.

[21] 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*, 38(2):69–74, 2008.

[22] D. Merkel. Docker: lightweight linux containers for consistent development and deployment. *Linux Journal*, 2014(239):2, 2014.

[23] J. R. Rymer and J. Staten. The Forrester Wave: Enterprise Public Cloud Platforms, Q4 2014. *Intelligence (BI)*, 4, 2014.

[24] V. Sekar, S. Ratnasamy, M. K. Reiter, N. Egi, and G. Shi. The middlebox manifesto: enabling innovation in middlebox deployment. In *Hotnets '11*. ACM, 2011.

[25] 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. *SIGCOMM 2012*, 42(4):13–24, 2012.

[26] G. Wang and T. Ng. The Impact of Virtualization on Network Performance of Amazon EC2 Data Center. In *INFOCOM 2010*, pages 1–9, March 2010.