IEICE TRANS. COMMUN., VOL.E100–B, NO.11 NOVEMBER 2017

INVITED PAPER *Special Section on Network Virtualization, Network Softwarization, and Fusion Platform of Computing and Networking*

Network Function Virtualization: A Survey

Malathi VEERARAGHAVAN[†a)], ***Nonmember***, **Takehiro SATO**[††], ***Member***, **Molly BUCHANAN**[†], **Reza RAHIMI**[†], ***Nonmembers***, **Satoru OKAMOTO**[††], ***and*** **Naoaki YAMANAKA**[††], ***Fellows***

SUMMARY The objectives of this survey are to provide an in-depth coverage of a few selected research papers that have made significant contributions to the development of Network Function Virtualization (NFV), and to provide readers insights into the key advantages and disadvantages of NFV and Software Defined Networks (SDN) when compared to traditional networks. The research papers covered are classified into four categories: NFV Infrastructure (NFVI), Network Functions (NFs), Management And Network Orchestration (MANO), and service chaining. The NFVI papers describe "framework" software that implement common functions, such as dynamic scaling and load balancing, required by NF developers. Papers on NFs are classified as offering solutions for software switches or middleboxes. MANO papers covered in this survey are primarily on resource allocation (virtual network embedding), which is an orchestrator function. Finally, service chaining papers that offer examples and extensions are reviewed. Our conclusions are that with the current level of investment in NFV from cloud and Internet service providers, the promised cost savings are likely to be realized, though many challenges remain.

key words: *Network Function Virtualization, Software Defined Networks, Software switches, middleboxes, orchestrators*

1. Introduction

Network Function Virtualization (NFV) is a term used to represent the implementation of data-plane network functions in software that is executed on commodity hosts. The hypothesis is that NFV will incur lower capital expenditures (capex) and operating expenditures (opex) when compared to traditional switches/routers and middlebox appliances in which data-plane network functions are typically implemented in custom hardware.

Early interest in NFV was expressed by communication service providers, with many of them collaborating on a 2012 white paper [1]. Subsequently, a standardization group called Network Functions Virtualisation Industry Specification Group (NFV ISG) was created by ETSI.

The architectural components of NFV are: (i) Network Function Virtualization Infrastructure (NFVI), (ii) Network Functions (NFs), and (iii) Management And Network Orchestration (MANO). Infrastructure consists of hardware (a single computer or a compute cluster), and framework software, which offers functions that are commonly required by NFs, such as NF placement, dynamic scaling, etc. Data-plane network functions considered for software implementation in commodity hosts range from basic packet forwarding to complex middlebox functions such as intrusion prevention systems. When NFs are executed on Virtual Machines (VMs), they are referred to as Virtual Network Functions (VNFs). MANO components include management functions, such as Fault management, Configuration management, Accounting, Performance monitoring and Security (FCAPS), and orchestrators, which manage service chains of multiple NFs.

Early research work on NFV focused on NF placement, also referred to as resource allocation, for optimally locating NFs in physical servers and/or VMs. A 2013 survey [2] focused on virtual network embedding, and reported on several optimization techniques for ideal NF placement.

A 2015 paper [3] on NFV provides an excellent survey, covering standards, industry NFV efforts, and research papers published before Feb. 2015. Therefore, this survey paper focuses primarily on research papers published after this date.

Another 2015 survey [4] focuses on software-defined NFV and service chaining. It provides the following comprehensive list of network functions suitable for NFV: (i) network switching elements such as broadband remote access server, broadband network gateways, and IP routers, (ii) mobile network systems, such as LTE eNodeB, and gateways, (iii) residential modems and routers, (iv) tunneling gateways, such as IPsec/SSL virtual private network gateways, (iv) traffic analysis elements for quality of experience measurement, (v) Service Level Agreement (SLA) monitors, (vi) Voice-over-IP systems such as IP multimedia sub-systems, (vi) application-level optimizers such as CDN servers, load balancers, and application accelerators and (vii) network security devices, such as firewalls, Deep-Packet Inspection (DPI) based and anomaly based Intrusion Detection System (IDS), denial-of-service attack detectors, and malware and spam detectors [4].

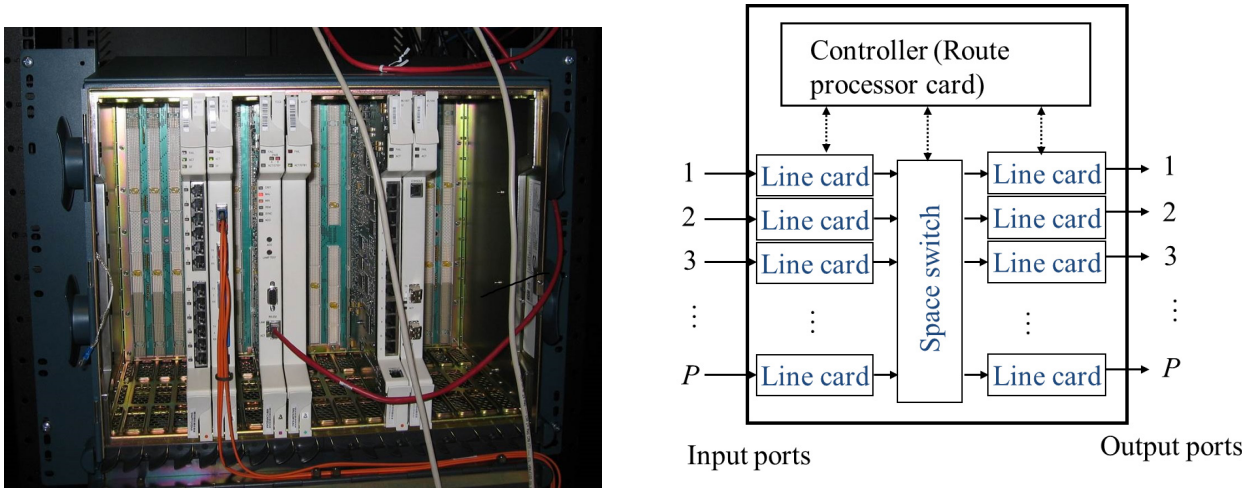
A 2016 NFV-security survey [5] organized challenges into the following categories: (i) NFVI security, (ii) defining interface standards for security functions, (iii) MANO security, and (iv) security challenges stemming from the dynamic scalability (elasticity) aspect of NFV. The paper then describes various security platforms that are both open-source and offered by companies.

The intent of this survey is to provide an in-depth treat-

Manuscript received April 10, 2017.
Manuscript publicized May 16, 2017.

[†]The authors are with University of Virginia, 351 McCormick Road, Charlottesville, VA 22904-4743, USA.
[††]The authors are with Keio University, Yokohama-shi, 223-8522 Japan.
a) E-mail: mv5g@eservices.virginia.edu

DOI: 10.1587/transcom.2016NNI0001



(a) Modular router

(b) Unfolded view of the router

Fig. 1 Router architecture.

ment of a few research papers, rather than a shallow coverage of a broad range of papers. Therefore, this survey is not comprehensive, but instead offers readers a fair understanding of the covered research papers.

Section 2 describes how NFV relates to traditional networks and Software Defined Networks (SDN). Section 3 describes three papers that offer NFVI framework software, useful to NF developers and service-chain creators. Section 4 has two subsections for NFs, one for software switches/routers, and another for middleboxes. Section 5 offers a light coverage of NFV management and orchestration systems. Section 6 surveys papers that offer solutions to service chaining. The paper is concluded in Sect. 7.

2. Background

2.1 Traditional Architecture

Figure 1(a) shows a photograph of a modular router, which includes a 12-port Ethernet line card, and a 4-port optical line card, a route-processor card with a single Ethernet control-plane port, and other cards. Figure 1(b) shows an unfolded view of the router with line cards being split into an input-side line card and output-side line card (in reality, input-side and output-side functions are on the same line card). The route-processor card, as the name suggests, has a built in processor, which runs an operating system and software that implements control-plane protocols, such as routing protocols and signaling protocols. Data-plane functions, primarily packet forwarding, are implemented in the line-cards. In high-speed routers[†], these functions are implemented in custom hardware, such as ASICs and Ternary CAMs (TCAMs).

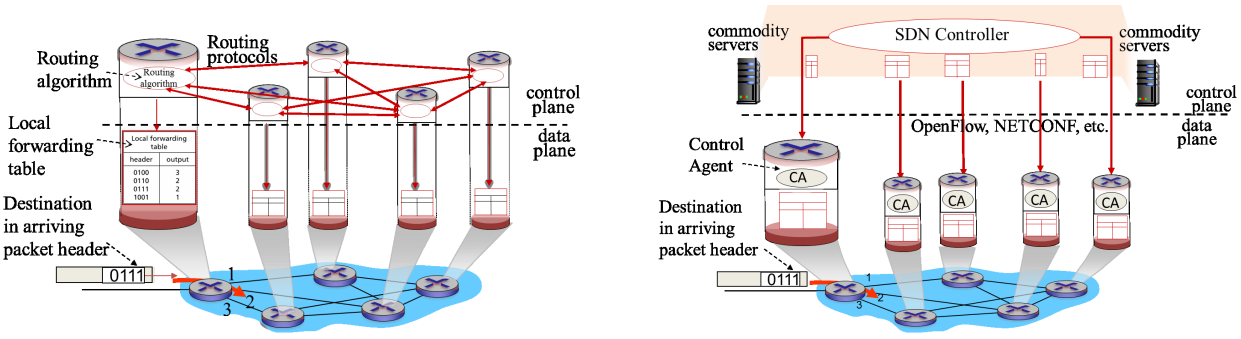
[†]We use the terms "routers" and "switches" interchangeably in this context because most high-end systems offer both L3 and L2 packet-forwarding services.

2.2 Software Defined Networks

When datacenters started deploying large numbers of computers, e.g., 1M computers in one datacenter, the cost of switches to interconnect these computers became significant. For example, if 48-port switches are used, more than 20K switches are needed to interconnect 1M computers. If each switch costs $100K (which is the cost of a 48-port 10GE switch [6]), the total capital expenditures for switching is significant. The high cost of these switches is attributed to software and hardware development costs. A significant component of opex cost is per-router software licensing and maintenance fees.

In Software Defined Networks (SDNs), most of the control-plane software is removed from the route-processor card, and instead re-implemented for execution on external commodity hosts, which are under the control of the cloud/Internet service provider. The SDN controller software can hence be maintained (modified for bug fixes) and upgraded to support new features by the provider itself, without having to wait for the switch vendor to make modifications. The hypothesis is that software development and maintenance costs in the SDN model will be lower than the difference in cost between a traditional switch/router model and a model designed for SDN networks.

Figure 2(a) shows a network of traditional routers, in which the control-plane protocols (such as routing protocols) and forwarding-table computation algorithms are implemented in software in the route-processor card as illustrated in Fig. 1. For example, Open Shortest Path First (OSPF) routing protocol modules in the routers will exchange messages with neighbors, learn the topology, run Dijkstra's or other shortest-path algorithms to compute the routing tables. The routing tables are then copied from the router processor card into forwarding tables stored in line cards. These forwarding tables are consulted by the ASICs/other hardware



(a) Traditional: Control-plane software implemented in each router

(b) Software Defined Network (SDN): A remote SDN controller implements the control-plane algorithms

Fig. 2 Illustration of the SDN concept [7].

that perform line-rate packet forwarding.

Figure 2(b) illustrates the SDN solution. In SDN, the cost of routers is lowered by removing the control-plane software from the router-processor cards, and instead running a simple control agent, e.g., OpenFlow agent, in the processors located within the routers/switches. Control-plane algorithms for actions such as forwarding-table computation are executed in SDN controllers installed on off-the-shelf computers as illustrated in Fig. 2(b). The computed forwarding tables are then downloaded to the routers/switches via a protocol such as OpenFlow or NETCONF.

Clearly, there cannot be a single giant SDN controller for the entire Internet! Therefore, each organization (domain or autonomous system) that runs its own network will deploy and operate an SDN controller. "East-West protocols" is the term used for inter-SDN controller communications, while "North-South protocols" is the term used for protocols between an SDN controller and the domain's own switches/routers.

In *summary*, the capex/opex cost of the routers/switches will be lower in the SDN solution since this equipment will not have the complex software required for control-plane protocols such as OSPF, BGP, etc. Some large cloud and service providers will be able to develop and maintain the SDN controller software at low costs, but this may be challenging for others, such as enterprise network organizations.

2.3 Network Function Virtualization

In Sect. 2.2, we noted that the high cost of switches/routers is largely attributable to software and hardware development costs. SDN offers a solution for reducing development costs of the software run on route-processors in switches/routers. NFV offers a solution for reducing hardware development costs of switches/routers. By observing that basic data-plane functions such as packet header lookup and forwarding can be implemented in software on a commodity server, the assumption that custom hardware is required for data-plane functions has been challenged by NFV.

Commodity x86/IA (Intel Architecture)-based servers are cost-effective and energy-efficient. Furthermore, new

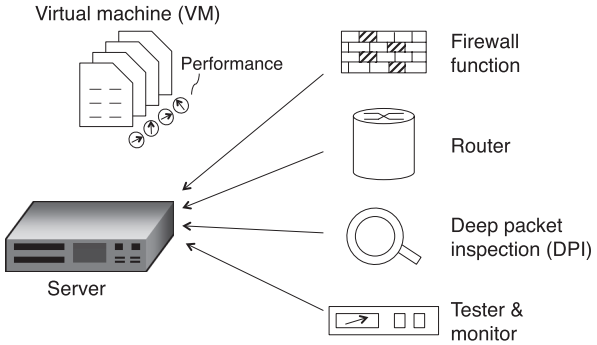


Fig. 3 Network Functions Virtualization (NFV) supports the execution of multiple network functions in one server.

servers, which implement the latest most-advanced processor, memory and disk technologies, and are more energy-efficient, can replace the old servers on which NFs are being executed. In addition, a software NF can be developed to run on multiple operating systems, VM hypervisors, and containers. Industry competition in the commodity-server and OS/hypervisor/container markets is expected to help reduce capex and opex of NFV-based network switches and middleboxes.

Figure 3 illustrates that using VMs, multiple network functions, such as firewall, router, DPI and tester/monitor, can be executed on a single server. Sharing the same physical server for multiple network functions allows for higher CPU resource utilization and energy efficiency when compared to the traditional solution in which each network function is implemented in its own appliance. In small campus networks, it is potentially feasible to implement all required network functions in a small number of physical servers using multiple VMs per server.

Figure 4 shows that in the traditional approach, each network appliance consumes a fixed amount of power, e.g., the firewall consumes 200 W, the router consumes 1.5 KW, and the tester/monitor consumes 400 W. In contrast, in NFV, if all three functions can be implemented in VMs in a single commodity server, there can be significant energy savings since such servers typically consume around 750 W.

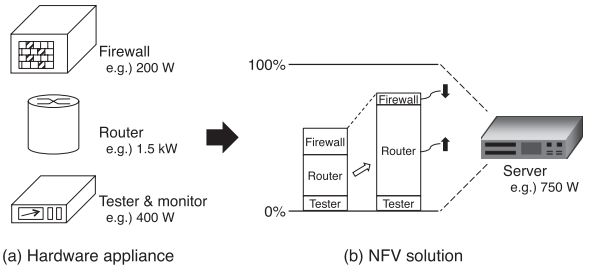


Fig. 4 Custom-hardware appliances consume more power than an energy-efficient NFV solution.

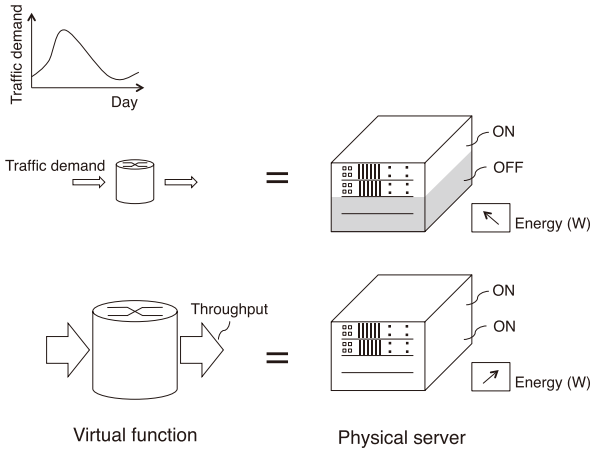


Fig. 5 NFV supports scalability with traffic load, while saving energy.

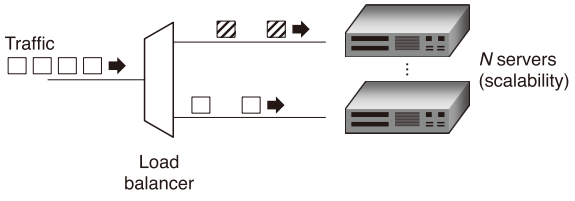


Fig. 6 Role of load balancer in NFV.

Figure 5 illustrates the scalability value of NFV. As traffic load increases, more servers can be powered-on, and during intervals of low traffic load, some servers can be powered off to save energy.

Figure 6 illustrates the role of a load balancer. A function of a load-balancer is to distribute incoming packets to different VMs and/or different servers. For example, in a firewall, incoming packets are distributed to different servers by hashing on fields in the packet header. Load balancing is required to support scalability.

Finally, Fig. 7 illustrates the pipelining approach to scalability, using IP router as an example. The IP router network function is divided into several sub-function blocks, and blocks are connected in a pipeline. In Fig. 7, four types of blocks are illustrated in a 4-stage pipeline. Block B is a longest-prefix matching engine (LME). Since longest-prefix matching is a complex operation, and therefore, two instances of Block B are executed in parallel to reduce packet-

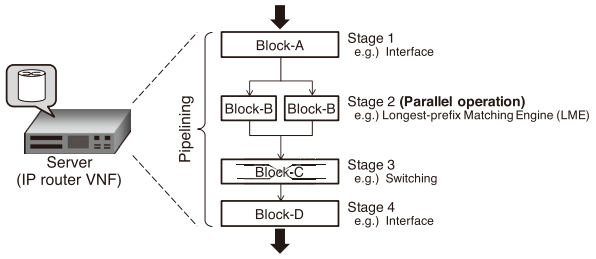


Fig. 7 Scale-up technique using pipelining method.

processing delay through this stage of the pipeline. In the pipelining approach, delay in each stage should be almost the same so that all CPU-cores are kept busy with a steady flow of packets. The pipelining approach is suitable for network functions if there are no loops between the blocks, e.g., block A is revisited after block C.

3. NFVI Framework

NFV infrastructure includes hardware and software. Since the hardware is primarily commodity servers, we focus here on NFVI software. The term "framework" has been used by some research papers to describe common software that is useful for NF developers. Functions such as NF placement, dynamic scaling, fault tolerance, and load balancing are examples of these common functions. Thus NFVI framework software will enable an NF, or a service chain consisting of multiple NFs, to achieve high performance, high reliability, security, etc.

In this section, we introduce three recently developed NFVI frameworks: ClickOS [8], [9], Elastic Edge (E2) [10], and NetVM/OpenNetVM [11], [12]. All these frameworks enable NFs to achieve near-line rates at 10 Gbps on commodity servers by optimizing I/O subsystems and using high-performance packet-processing libraries. ClickOS and NetVM/OpenNetVM are frameworks designed for a single server, but support scalability by increasing the number of VMs or server components (i.e., CPU, memory, and NIC). E2 supports clustered servers connected by switches, and supports scalability by increasing the number of physical servers (or racks). Support for service chaining is available on all these frameworks in either static or dynamic mode.

3.1 ClickOS

Motivation:

ClickOS [8], [9] is a VM platform that supports a variety of middlebox functions. Current custom-hardware based middleboxes have problems such as high costs and management issues, and are typically inflexible in their support for modifications. Software-based middleboxes, designed to run on commodity servers, are hence desirable. However, middlebox functions run on common hypervisors, such as KVM and Xen, often suffer from low performance, since these hypervisors are not optimized to support VMs running

middlebox functions. ClickOS was developed to provide a common platform for software middleboxes based on Click software. It also addresses the problems of boot-up delays with VMs, high resource utilization, and low performance.

Solution:

ClickOS is a combination of a Xen-based OS called "MiniOS" and Click modular router [13], [14]. MiniOS offers all the functions needed to run Click, without the additional functions in a typical Linux kernel. For example, Linux-kernel functions such as multi-user support, support for multiple memory address spaces, and support for filesystems and other devices, are not required to run Click. Thus ClickOS saves CPU cycles and memory, and has a short boot-up time. Each instance of ClickOS run on a physical machine is essentially a VM. Users can create and destroy ClickOS VMs by using the CLI and connecting to a Xenstore database that manages control threads of MiniOS. In addition, a network driver was optimized by modifying its receive function, and using a virtual network device and software switch based on netmap [15]. ClickOS achieves high scalability by running many lightweight VMs, when compared to the current commonly used VM solutions. The ClickOS software is open-source and distributed under the 3-Clause BSD license [16].

Evaluation:

The authors evaluated the performance of ClickOS on x86 commodity servers (one server was used to run a packet generator and a packet sink, and another server was used for ClickOS VMs). Even though ClickOS was compiled with more than 200 Click elements, the size of the ClickOS VM image was as small as 5 MB. A ClickOS can be created and initiated to run middlebox functions within approximately 30 milliseconds. The authors successfully ran 128 ClickOS VMs on a single server and demonstrated nearly 10 Gbps throughput on a single 10GE NIC. By using additional CPUs and 10GE NICs, ClickOS achieved 27.5-Gbps packet-forwarding performance on a commodity server. The paper also described implementations of a few middlebox functions (such as firewall, NAT, and load balancer), and reported that ClickOS achieved almost line-rate processing when the packet size was 512 bytes or larger.

3.2 Elastic Edge (E2)

Motivation:

A framework called Elastic Edge (E2) offers NF developers a set of solutions for common problems such as dynamic scaling, and NF placement [10], and thus saves each programmer the effort of having to implement code to deal with these issues. It can be used for a single NF implementation or an NF service chain. The solution draws lessons from the MapReduce framework, which "hides the messy details of parallelization, fault-tolerance, data distribution and load balancing" [17] from programmers in the data-analytics space.

Solution:

The E2 software consists of three components: (i) E2 Manager that monitors and enables the creation and deletion of NFs (which can be programmed by others) on a set of servers, (ii) E2 Dataplane (E2D) that provides a set of libraries for use by NFs, and a (iii) Server agent that manages the NFs on each server. Through an E2 interface, an operator specifies NF descriptions and the underlying hardware (e.g., number of cores per server). This interface allows the operator to specify a set of policy statements called pipelets. A pipelet is defined for each traffic class. The pipelet specifies a directed acyclic graph (DAG) for each traffic class, which shows how packets from the corresponding traffic class are processed by NFs. The E2D software is an enhancement of a package called SoftNIC, which runs on top of Intel's Data Plane Development Kit (DPDK) [18]. SoftNIC leverages features of DPDK such as polling for packets for high-speed implementation of packet-processing functions. An example E2D extension to SoftNIC consists of load monitoring and load balancing between NFs (necessary to support dynamic scaling). The E2 Control plane parses the input about NFs and hardware provided by an operator and determines on which server to place each NF, how to interconnect the NFs to realize services, how to dynamically scale the hardware resources used as traffic load changes, and ensures host affinity (which means all packets of a flow are sent to the same NF instance). The E2 software is open-source [19].

Evaluation:

The E2 prototype consists of off-the-shelf servers interconnected by commodity switches with N ports, of which K ports carry traffic in and out of the E2 cluster while the rest of the ports are used for inter-NF (intra-E2-cluster) traffic. The E2 solution is evaluated to determine whether the E2D layer adds a significant latency overhead, and whether it affects throughput. The results show that a solution in which the NF is executed directly over DPDK vs. a solution with E2D in between the NF and DPDK have almost the same performance (latency impact is less than a μs). Experiments were run to measure the performance of three NFs: an IDS, a URL-based filter, and a WAN optimizer, with and without the bytestream vports feature of E2. This feature performs the common function of TCP flow reassembly required in DPI based NFs. If there are multiple such NFs in a service chain, the bytestream vports feature saves processing capacity.

Impact:

The paper uses the drive by telecommunication carriers to modernize Central Offices (COs) as a motivation for E2. COs support many functions, such as DPI, NAT, DDoS prevention, firewalls, WAN acceleration, etc. Therefore, with a solution such as E2, a service chain of NFs running on multiple servers, can be created and managed with an E2 manager.

3.3 NetVM and OpenNetVM

Motivation:

The goal of NetVM [11] was to provide an NFV platform that can provision network functions on commodity servers at high throughput (near line speed for 10GE links). The authors note that while SDN has enabled dynamic and flexible configuration in the control plane, data-plane NF implementations are still limited in performance.

Solution:

NetVM is an NFV environment built over the KVM platform and the Intel DPDK library. In the data plane, network functions (e.g., firewall, proxy, router) are embedded within VMs. Network services are composed by pipelining VMs. To leverage the performance enhancements offered by DPDK, NetVM applies a shared-memory framework that provides packet delivery between VMs without copying. NetVM also supports security domains so that packets are processed only in trusted VMs. Packet flows and VMs in the NetVM platform are managed by the tight combination of an OpenFlow controller and an NFV orchestrator.

Evaluation:

The authors evaluated NetVM on two commodity servers, which were used to run a traffic generator and the NetVM system. A NetVM L3 packet forwarding engine achieved the full line rate of a 10GE NIC, while other solutions (e.g., Click, SR-IOV [20]) achieved only 5–6 Gbps on the same servers. The authors also investigated the scalability of NetVM by increasing the number of CPUs and the number of ports. With another server, which had 28 cores and 4 NICs, a NetVM L2 packet forwarding application achieved 34.5 Gbps, while SR-IOV achieved only 22 Gbps.

Extension:

Based on the architecture of NetVM, OpenNetVM [12] was developed by the same team. In OpenNetVM, network functions are run in Docker containers instead of KVM VMs. This container-based architecture offers a lighter and simpler deployment of network functions, and allows for easy replication of network functions for scalability. Both the management framework and network functions have control capabilities, which enable OpenNetVM to configure service chains dynamically. The authors evaluated their service-chaining solution with an experiment, which showed that OpenNetVM dropped about 4% of the packets in a 6-function chain, while ClickOS dropped about 39% of the packets in a 3-function chain. The paper also reported that the time to initiate network functions in a container is only 0.526 seconds on average, while the same functions required at least 12 sec when run on KVM VMs in the original NetVM platform. The OpenNetVM software is open-source and distributed under the BSD license [21].

4. Network Functions

Section 4.1 describes software switches/routers that perform packet forwarding functions. Section 4.2 describes two middlebox implementations.

4.1 Software Switches/Routers

Before reviewing new research papers on software switches, we note that software switches are designed for two purposes: (i) to support packet forwarding between VMs within a server, and (ii) as a replacement for physical switches that interconnect servers. Solutions in the first category include Open vSwitch (OVS) [22], a hypervisor offload solution [23], and SnabbSwitch [24]. Solutions in the second category include the Click modular router [13], RouteBricks [25], PacketShader [26], Lagopus [27], [28] and ScaleBricks [29].

Sections 4.1.1, 4.1.2 and 4.1.3 describe the three solutions in the first category. Section 4.1.4 provides a short review of three pre-2015 solutions, Click, RouteBricks and PacketShader. Section 4.1.5 reviews netmap and DPDK, both of which offer libraries and packages for fast packet processing. Such libraries are essential for packet processing at high speeds for solutions in the second category. Sections 4.1.6 and 4.1.7 describe two solutions, Lagopus and ScaleBricks, both of which use DPDK.

4.1.1 Open vSwitch (OVS)

Problem:

When datacenters embraced virtualization, datacenter networking had to change to support a growing number of virtual ports, i.e., ports corresponding to virtual machines. Early virtual switches within the hypervisor connected virtual machines directly to the physical L2 switches. This approach is not scalable because every time a new set of VMs is created, the physical L2 network switches have to be reconfigured. Furthermore when VMs are migrated, the physical L2 switch tables need to be modified by a controller or pay the cost of address learning. Therefore, scalability and mobility were the primary drivers for a new type of virtual switch. Virtual switch software provides packet forwarding between VMs, while physical switches are used only for forwarding packets between hypervisors (effectively, between VMs on different physical servers).

Solution:

OVS [22] comprises a user-space daemon that can be run on different operating systems (OS), and a data-path kernel module that is OS-specific. When a packet arrives, it first goes through the kernel module. If the kernel module has instructions on how to deal with the packet, the packet is handled according to the instructions. If not, the kernel module sends the packet to the user-space daemon, which handles the packet, creates instructions for future handling of similar packets, and sends both the packet and the instructions back to the kernel module. The OVS user-space daemon receives and processes OpenFlow messages sent by an external controller to create the packet-processing instructions. But since OpenFlow does not support certain functions, such

as configuring queues for QoS support, an OVS DataBase (OVSDB) management protocol has been defined and implemented as a complement to OpenFlow. OVS maintains both flexibility and high performance without excessive use of hypervisor resources through the use of flow caching (for microflows and megaflows), a packet classifier that uses tuple space search [30] and is caching-aware, and proactive flow-table programming. Cache invalidation is distributed across multiple threads to increase performance and to prevent revalidation operations from blocking the setup of new flow entries. In short, this 2015 paper [22] describes many improvements made to OVS. The OVS software is open-source and distributed under the Apache License 2.0 [31].

Evaluation:

Operational performance of OVS in a commercial datacenter was characterized. The authors recorded 24 hours of OVS performance data, with statistics on cache size, cache hit rate, and CPU usage, by polling more than 1000 hypervisors every 10 minutes. Observed cache sizes were generally small, and even the largest was well within the limit set by OVS. Hit rates were high when traffic was heavy (98.0%) and lower under a lighter load (74.7%), with an overall value of 97.7%. Few hypervisors saw large traffic loads. CPU load for the OVS user-space daemon was, on average, 5% or less for 80% of hypervisors. In addition, experiments were executed to characterize caching performance.

4.1.2 Hypervisor Offload Solution

A 2016 paper by Wang et al. [23] first lists various host virtualization solutions (KVM, Xen, VMware, and containers) and I/O virtualization techniques (bridge, virtio, Passthrough and single-root input/output virtualization (SR-IOV)). The authors then describe their implementation of a simple L2 forwarding engine, and an L3 forwarding software using CuckooSwitch [32], [33] and DPDK. The hypervisor was KVM and I/O virtualization techniqe was SR-IOV with Passthrough. An extensive set of experiments was executed with two hosts that were interconnected via two 10GE links. A high-speed traffic generator (`pktgen-dpdk`) was run on one host, and L2-forwarding or L3-forwarding software was run on the second host. The experiments compared packet-forwarding throughput, latency and jitter when using a random assignment of packets to VMs (running on lcores; each server had 32 lcores), a NUMA-aware assignment of packets to VMs, a worst-case assignment, and a bare-metal test. Measurements showed that bare-metal outperformed any configuration in which VMs were used.

Noting that virtualization overhead will limit NFV scaling, the authors looked for a better solution to support high-speed packet forwarding between VMs within a virtualized server. The authors developed a solution in which data-plane packet forwarding is offloaded to the hypervisor. The authors state that their solution is similar to OVS, which as described above, has a kernel module for packet forwarding. The paper shows experimental results that attest to better throughput and latency performance for the hypervisor offload solution when compared to OVS-DPDK (which is discussed in Sect. 4.1.5).

4.1.3 SnabbSwitch

SnabbSwitch [24] is an open-source user-space virtual switch for the KVM hypervisor, implemented in Lua, which is a scripting language that was proposed in 1993 and has evolved to support applications in many domains [34]. It adds two innovations to the field of virtual switches: LuaJIT, which is a high-performance implementation of Lua, and vhost-user, which allows VMs to send network traffic directly to a user-space virtual switch such as SnabbSwitch, without passing through the kernel. SnabbSwitch comprises three parts. First, `app` is sofware that implements a network function, or core functions, such as a NIC driver and `virtio-net` device, which allows NFs running in VMs to send traffic directly to SnabbSwitch, bypassing the kernel. Apps are the means by which SnabbSwitch can be extended and altered (the authors have already added a traffic limiter and a packet filter). Second, a `link` is a buffer that holds packets between different `apps`. Third, the `app engine` configures, initializes, and manages `apps` and `links` to control the flow of execution.

After optimizing SnabbSwitch, its performance was compared with that of Virtual Function IO (VFIO) [35] and SR-IOV [20], both of which use hardware assistance, Linux Bridge, OVS, and OVS-DPDK for two scenarios: (i) unidirectional VM-to-VM communication, and (ii) bidirectional VM-to-VM communication in which the L2FWD application provided by the DPDK open-source project was run as a guest in its own VM. Results show that SnabbSwitch can perform packet forwarding between VMs at multiple Gb/s, and its relative performance to other solutions depends upon the tested scenario and packet size. SnabbSwitch software is open-source and distributed under the Apache License 2.0 [36].

4.1.4 Click, RouteBricks, PacketShader

Given our focus on post-2015 research papers, in this section, we note, very briefly, a few interesting points about three seminal papers on software switches. These include the 2000 Click modular router paper [13], the 2009 RouteBricks paper [25] and the 2010 PacketShader paper [26].

Since the *Click modular router* work predated, by many years, the NFV movement, we were interested in understanding the motivation for Click. The motivation was primarily to support research on active topics under investigation such as packet dropping policies, differentiated services, etc. The Click modular router is open-source software [14]. *RouteBricks* uses Click software and achieves high speeds by using multiple cores within a single server, and multiple servers interconnected in a generalized butterfly topology. With 4 servers, RouteBricks processed 64B packets at 12 Gbps. The RouteBricks software is open-source and distributed under

the GNU GPL v2 [37]. *PacketShader* is a software router designed to run on a system with GPU accelerators. PacketShader, running on a single commodity host, processed 64B packets at 39 Gbps. The PacketShader software is open-source and distributed under the GNU GPL v2 [38].

4.1.5 Libraries for Fast Packet Handling

Support for fast packet processing is provided by solutions such as netmap [15] and Intel's DPDK [18]. For example, Rizzo states that "netmap is a framework to reduce the cost of moving traffic between the hardware and the host stack" [15]. As noted in the web site [39], netmap is a kernel module, which has been implemented in Linux, FreeBSD, and Windows. Unlike with DPDK, there are no restrictions on the NIC hardware that can be used with netmap. A companion software switch called VALE is also described in the netmap web site. The netmap software is open-source and distributed under the BSD 2-clause "simplified" license [39].

DPDK is a user-space implementation available for FreeBSD and various Linux implementations [40] and requires specific types of DPDK-compatible NICs [41]. DPDK supports polling, packet coalescence, lock-free queues, the use of huge pages, and zero-copy transfers. When compared to interrupt mode, polling is better for achieving lower response times. Lock-free queues are an important mutual exclusion solution necessary for high-performance systems [42]. Lock-free queues are used between I/O threads and packet-processing threads. The use of huge pages to reduce the rate of Translation Lookaside Buffer (TLB) misses is important in packet processing given that most tasks involve memory accesses. Finally, the data direct I/O technology [43] provided by DPDK moves data directly from the NIC to the last-level processor cache, and thus reduces main-memory accesses. DPDK is open-source and distributed under the 3-Clause BSD license [18].

A version of OVS ported to DPDK is available on GitHub [44], though this software has not been modified since 2015. A Dec. 2016 version is reported by Intel [45], and RedHat [46].

4.1.6 Lagopus

Lagopus [27], [28] is a software OpenFlow switch designed for wide-area network service providers. Lagopus is designed to be run on multi-core processors. For example, in an eight-core prototype, two cores were used for I/O receive threads, two cores for I/O transmit threads, and four cores for flow-table lookup worker threads. The receive and transmit threads use DPDK libraries and drivers to move packets in and out of NICs. Ring buffers are shared between the I/O receive thread and the worker thread on the ingress side, and between the worker thread and the I/O transmit thread on the egress side. Lagopus uses parallelization rather than pipelining. In other words, all functions executed on a single packet header are handled by the same worker thread, while different packets are sent to different worker threads. Further, the Lagopus implementation includes: (i) packet classification for load balancing with explicit assignment of packets to worker threads, (ii) batch handling of packets (packet coalescing) between the I/O threads and worker threads, and (iii) high-performance flow-table lookup solutions [47]. Packet forwarding rates of 10 Gbps were reported in experimental studies of Lagopus performance [27], [28]. The Lagopus software is open-source and distributed under the Apache License 2.0 [48].

4.1.7 ScaleBricks

ScaleBricks [29] offers a design for creating network appliances by using clusters of servers. Unlike RouteBricks, ScaleBricks uses a switch for load balancing. The primary goal of ScaleBricks is to offer throughput scaling, scaling of the forwarding table (also called Forwarding Information Base, or FIB) and scaling of the FIB-update rate. It builds on the authors' prior work on CuckooSwitch mentioned earlier, and it uses DPDK. A ScaleBricks design consisting of N servers uses a switch to interconnect these servers. For fast packet forwarding, the whole FIB should be located in each server so that the ingress server can lookup the FIB to determine the egress server for each arriving packet. However, this design requires a large amount of memory, and a high update rate as all servers must store the whole FIB. Another design is to divide the FIB into N partitions, and have the ingress node determine the indirect node that holds the appropriate partition for an arriving packet by using a hashing scheme. The indirect node would then be able to determine the egress node (from its FIB partition) to which an arriving packet should be directed. However, this scheme has the disadvantage of two-hop latency. The key contribution of ScaleBricks is a solution that avoids both the two-hop latency required in the hash-partitioned design, and the need for large memory required in the solution in which the full FIB table is located at each server.

Packet forwarding in an LTE-to-Internet gateway is the application selected for testing ScaleBricks. Significant performance improvements of a commercial gateway were demonstrated with the ScaleBricks design. This paper discusses problems such as skewed forwarding table distribution between the servers (which can happen because a controller assigns flows to different servers), failure handling, and states the authors' goal to test the ScaleBricks design for other stateful applications.

4.2 Middleboxes

As examples, two research papers that describe implementation of middleboxes in software for commodity hosts are described in depth. Section 4.2.1 describes a firewall framework implementation, and Sect. 4.2.2 describes a NAT NF implementation.

4.2.1 Firewall

Problem:

Deng et al. [49] address the security challenge of providing firewall protection for virtual networks, in which network functions are implemented in VMs on servers. The challenge lies in protecting dynamically changing network topologies and perimeters, which occurs due to VM migration and automated scaling of NFs. Traditional firewalls – with their dependence on restricted entry points to a more static network topology – cannot meet the requirement of defending virtual networks.

Solution:

The proposed solution is a framework called VNGuard, designed specifically to manage virtual firewalls on virtual networks. The three major contributions are: (i) a high-level language for specifying security policies, (ii) a method for computing an optimal placement of virtual firewalls, and (iii) means for adapting virtual firewalls to protect the virtual network as its VMs change and/or migrate. Therefore, the main contribution is not an implementation of a specific firewall NF, but rather a framework for supporting software-based firewalls. To evaluate VNGuard, the authors implemented a firewall using ClickOS [9] as a base, and all the components of the VNGuard framework. As ClickOS requires that a virtual firewall be rebooted every time its rules are updated, the authors designed three new Click elements: to store firewall rules, process packets against the rules, and support firewall rule additions/deletions. VNGuard software is currently a research prototype, but Deng et al. [49] plan to implement VNGuard in open-source NFV platforms such as Open Platform for NFV (OPNFV) [50].

Evaluation:

Using NSF CloudLab testbed, three experiments were run to: (i) evaluate firewall performance, (ii) test the optimal firewall placement module, which used a Matlab Integer Programming solver, and (iii) evaluate adaptability by measuring the time to add/delete firewall rules. Firewall throughput was measured for fairly low incoming packet rates (10 Mbps to 90 Mbps) [51]. Improving packet processing rates was not a goal of this work. For a firewall with 900 rules (a 2012 reference [52] is cited to state that an average firewall has 793 rules), processing time was around 6 to 7 microseconds. Optimal firewall placement depends on the number of rules, and the number of instances (VMs). As both parameters increase, the time to find an optimal solution also increases. But even with 2100 rules across seven instances, it took less than two tenths of a second to place a firewall. Firewall rule addition took fewer than 300 ms to alter 450 rules; deletion speed was similar.

4.2.2 NAT

Olteanu et al. [53] set out to determine whether commodity hosts running a software implementation of a stateful application such as NAT can achieve performance comparable to that of NAT appliances designed with custom hardware. The answer is that their NAT software when run on six commodity hosts could perform address translation at 40 Gbps with 64B packets. By adding 9 more servers, their NAT software could handle 64B packets at 100 Gbps. Such high-speed NAT systems are required for provider deployments, and are referred to as carrier-grade NAT.

Their carrier-grade NAT solution consists of three components: (i) high-performance NAT software for a single host, (ii) a load balancer that distributes packets to multiple hosts running the NAT software, and (iii) dynamic scaling, i.e., adding more NAT hosts as the load increases, and removing hosts as the load decreases.

The single-host NAT software leverages Click modular router and netmap software. A single-core instance of this NAT software can process 2M packets/sec. Their load balancer consists of an OpenFlow 1.0 switch that is preprogrammed with flow-table entries to distribute packets coming from inside the network as well as packets coming from outside the network efficiently. To keep the size of the flow table small, the solution requires the assignment of external IP addresses to each of the individual NAT host interfaces. Other solutions are possible if not limited by OpenFlow 1.0 features. For dynamic scaling, after discussing the overhead of VM migration, the solution is to migrate connections in groups, and to daisy-chain NAT processing in one of the original hosts and a new host while migrating connections.

5. Management and Network Orchestration (MANO)

Mijumbi et al. [54] describe NFV MANO as consisting of (i) Virtualized Infrastructure Manager (VIM), (ii) VNF managers, and (iii) NFV orchestrator. The VIM manages the physical servers and virtual machines that constitute the NFVI hardware. The VNF managers manage individual network functions. The NFV orchestrator supports the chaining of network functions to create services. This paper also describes several NFV MANO projects, and pre-standardization MANO products, that are interestingly from equipment vendors such as Alcatel-Lucent, Ciena and HP, not communication service providers. Finally, some of the MANO research challenges listed include resource management, FCAPS management across a service lifecycle, programmability, and interfacing.

The relation between the components of the MANO architecture and traditional Operations Support Systems (OSS) and Business Support Systems (BSS), and the interface specification between the NFV Orchestrator and OSS/BSS, are described in an ETSI document [55].

Although the term MANO does not appear in a paper [56] on an SDN-NFV architecture, the paper describes a hierarchical solution that includes an SDN controller, a service manager, an NF orchestrator and NF managers. Protocols are defined between the various components. Solutions are

provided for creating static and dynamic service chains. The solution is implemented and evaluated. The goal is to reduce load on the SDN controller. This architecture appears to inter-mingle control-plane and management-plane functions, but this inevitably occurs when considering configuration management.

One of the main problems in service chaining is resource allocation. Since an orchestrator manages service chains, one of its fundamental roles is to perform optimal resource allocation. A comprehensive survey [57] reviews papers on resource allocation. For example, Xia et al. [58] focused on a service chaining problem in an electrical/optical hybrid data center. They propose a heuristic algorithm for on-demand VNF placement that minimizes the number of optical/electrical/optical (O/E/O) conversions. In another paper [59], the same authors proposed using NF service chains to detect and steer large flows to the optical domain, and quantify the resulting energy savings possible with this solution. Bari et al. [60] proposed a heuristic algorithm for enterprise networks that finds the appropriate number and placement of VNFs to optimize opex and network resource utilization without violating SLAs. D'Oro et al. [61] applied non-cooperative game theory to compose a service chain with distributed computing. The proposed scheme converges to a Nash equilibrium in polynomial time, and preserves network users' privacy. Qu et al. [62] proposed a VNF scheduling method based on a genetic algorithm that reduces makespan and latency.

6. Service Chaining

Section 6.1 offers a general example of service chaining. Section 6.2 describes an example service chain created with ApplianceBricks. Section 6.3 describes two ubiquitous grid (uGrid) service chain examples. Atomic functions and disaggregation are concepts related to service chains and are hence described in Sect. 6.4. All the software described in this section are research prototypes, and do not appear to be available as open-source packages.

6.1 General Example

The term service function chaining (or simply service chaining) denotes "an ordered list of instances of service functions" (such as firewalls, load balancers, NATs, or other application-specific functions), and "the subsequent 'steering' of traffic flows through those service function" [63], [64].

Figure 8 illustrates a service chain between two routers. The service chain consists of a firewall, a NAT, a DPI, and a cache, all of which are implemented in software on commodity hosts. These network functions could be deployed in VMs in a single server, or could be distributed across multiple servers. The servers could be in an edge-cloud or in a commercial-cloud datacenter. High-speed optical networks offer the opportunity to offload some NFs to a remote datacenter if additional compute resources are required. However, WAN propagation delays may impact the overall service-chain performance. Therefore, performance requirements should be considered while choosing servers, and their corresponding datacenters, for execution of NFs in a service chain.

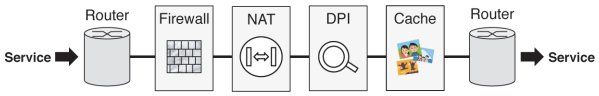


Fig. 8 Service chaining for Internet service.

Multiple service chains could share one network function. For example, Deep Packet Inspection (DPI) is a complex function that is not executed on all flows in one service chain. Therefore, a DPI module could be part of multiple service chains.

6.2 ApplianceBricks

The ApplianceBricks solution [65] is positioned as a general architecture for NFV. The implementation uses Click and DPDK. A 4-server cluster prototype is used to test NF service chains consisting for three applications: (i) mini-forwarding, (ii) IP-layer forwarding, and (iii) firewall. Mini-forwarding is basic packet forwarding from an input interface to a specified output interface without header processing [66]. In the first two service chain configurations, only mini-forwarding application is tested and measurements show a throughput of 6.8 Gbps with 64B packets. The third service chain includes 5-tuple dispatching, firewall filtering, and IP-layer forwarding, and a performance rate of 2.2 Gbps with 64B packets is reported.

6.3 uGrid Service Chain Examples

The uGrid environment [67], [68] extends the NFV service chaining concept from network functions to devices, general-purpose software program, and content. These additional components are referred to as "service parts."

Figure 9 shows an example of service chains in the uGrid environment. The bold dashed line shows a service chain consisting of (i) a camera (device), (ii) a video-processing software program running on an edge server, (iii) a program running on a commercial cloud-computing server, which could, for example, merge the camera video with some other related video content, and (iv) reception of the combined video on a television. The uGrid solution offers tools for creating such flexible service chains.

The narrow dashed line in Fig. 9 illustrates another service chain. This is a Machine-to-Machine (M2M) service chain in which an autonomous vehicle receives video data from an in-network video-processing program running on an edge server. Thus, the uGrid extended service-chain concept can be applied for such critical services in the future.

To create service chains and receive desired services in the uGrid environment, clients should first locate (discover) the required service parts. A method for discovering service

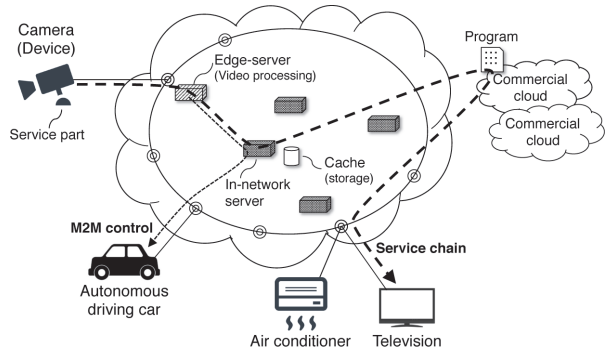


Fig. 9 Example service chains in Ubiquitous Grid (uGrid) networking.

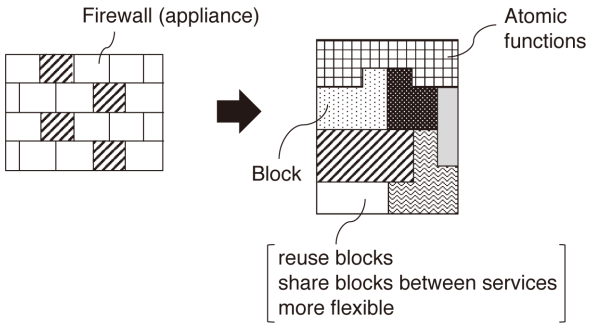


Fig. 10 ATOMIC-NFV: A firewall NF implemented with smaller blocks.

parts, based on Universal Description, Discovery and Integration (UDDI), was proposed by Akagi et al. [67]. Service parts are divided into multiple groups, and each group is managed by several UDDI registries. In each group, one of the UDDI registries is configured as a super-peer. The super-peer stores information about all the UDDI registries in the group. Search queries for service parts, which are sent by clients, are processed by multiple super-peers evenly to avoid query losses due to access concentration.

To realize the uGrid environment, another approach, called IP Routing/Signaling-based uGrid service provisioning, was proposed by Ishii et al. [68]. In this approach, IP addresses are assigned to each service part so that service chains can be provisioned in the network layer. The search for service parts, and the routing of service chains (called Service Routing), are executed with an extended version of Open Shortest Path First-Traffic Engineering (OSPF-TE), which advertises the link state of each service part, e.g., content type, current processing load, and current power consumption. Based on the topology of service parts determined by the extended OSPF-TE, the service-chain route is computed. The reservation of resources in service parts, and the establishment of a service chain (called Service Signaling), are executed with an extended version of Resource reSerVation Protocol-Traffic Engineering (RSVP-TE) of Generalized Multi-Protocol Label Switching (GMPLS). The extended RSVP-TE establishes the connections between service parts, and configures the function of each service part along the service-chain route. Ishii et al. implemented a prototype system for the provisioning of a simple video grid service using Service Signaling [68].

6.4 Atomic Functions and Disaggregation

Okamoto et al. [69] proposed to divide VNFs into small sub-functions called "atomic functions" as illustrated in Fig. 10. A network service with flexible performance can be provisioned by chaining atomic functions. The output of an atomic function could be transferred as input to another atomic function located in the same VM/server, or in another VM/server. The size of each atomic function is determined by the processing power of the CPU. The use of atomic functions increase the flexibility with which resources can be allocated.

The atomic-function concept is similar to the "disaggregation" concept proposed for optical transport networks [70], [71] and server equipment [72]. With disaggregation, hardware is grouped into resource pools [73]. Resources are chained together to realize a transport function or a computing function. A specific hardware component could be considered as an atomic function.

These concepts stretch the basic ideas of NFV and service chaining to include network transport hardware and server hardware, to offer a greater level of flexibility and scalability. To fully leverage these concepts, corresponding advances are required in SDN controllers and service-chaining orchestrators.

7. Conclusions: Pros and Cons of NFV

Early IP routers in the 70s were implemented in software on general-purpose computers. It has been interesting to observe how the progression of processing, memory, and cloud computing technologies has brought us full circle back to NFV. Whether NFV can deliver on its promise of saving capex and opex for service providers and enterprise networks remains to be seen. Open-source software is often viewed by IT division managers as requiring network engineers who are skilled software developers, which could increase opex. General-purpose hosts running common operating systems could be more vulnerable to security attacks when compared to proprietary router operating systems. Also, whether scalable-NFV clusters can handle 100-Gbps packet processing at lower capex when compared to traditional custom-hardware based routers is an important question to answer for service providers. In spite of these challenges, NFV offers academic researchers an exciting opportunity to experiment with new types of protocols, new techniques for fast packet processing, table-lookup operations, and for the marketplace to create and test new networking service ideas. Furthermore, many cloud and Internet service providers have invested considerable resources in developing NFV, and therefore the promised capex/opex savings are likely to be realized.

Acknowledgment

This work was supported by UVA NSF grants ACI-

1340910, CNS-1405171, and CNS-1531065. This work was also supported by Keio JSPS KAKENHI Grant Number JP16K00139.

References

[1] "Network functions virtualisation: An introduction, benefits, enablers, challenges & call for action." https://portal.etsi.org/nfv/nfv_white_paper.pdf
[2] A. Fischer, J.F. Botero, M.T. Beck, H. de Meer, and X. Hesselbach, "Virtual network embedding: A survey," IEEE Commun. Surveys Tuts., vol.15, no.4, pp.1888–1906, Fourth 2013.
[3] R. Mijumbi, J. Serrat, J.L. Gorricho, N. Bouten, F.D. Turck, and R. Boutaba, "Network function virtualization: State-of-the-art and research challenges," IEEE Commun. Surveys Tuts., vol.18, no.1, pp.236–262, Firstquarter 2016.
[4] Y. Li and M. Chen, "Software-defined network function virtualization: A survey," IEEE Access, vol.3, pp.2542–2553, 2015.
[5] W. Yang and C. Fung, "A survey on security in network functions virtualization," 2016 IEEE NetSoft Conference and Workshops (NetSoft), pp.15–19, June 2016.
[6] "L3 switch ApresiaNP7000-48X6L (48 10G SFP+, 6 40G QSFP+, OpenFlow1.3 ready)." http://www.apresia.jp/products/ent/np/series/core.html
[7] J.F. Kurose and K. Ross, Computer Networking: A Top-Down Approach Featuring the Internet, 2nd ed., Addison-Wesley Longman Publishing, Boston, MA, USA, 2002.
[8] J. Martins, M. Ahmed, C. Raiciu, and F. Huici, "Enabling fast, dynamic network processing with ClickOS," Proc. Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking, HotSDN'13, pp.67–72, ACM, New York, NY, USA, 2013.
[9] J. Martins, M. Ahmed, C. Raiciu, V. Olteanu, M. Honda, R. Bifulco, and F. Huici, "ClickOS and the art of network function virtualization," 11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14), pp.459–473, USENIX Association, Seattle, WA, 2014.
[10] S. Palkar, C. Lan, S. Han, K. Jang, A. Panda, S. Ratnasamy, L. Rizzo, and S. Shenker, "E2: A framework for NFV applications," Proc. 25th Symposium on Operating Systems Principles, SOSP'15, pp.121–136, ACM, 2015.
[11] J. Hwang, K.K. Ramakrishnan, and T. Wood, "NetVM: High performance and flexible networking using virtualization on commodity platforms," IEEE Trans. Netw. Serv. Manage., vol.12, no.1, pp.34–47, March 2015.
[12] W. Zhang, G. Liu, W. Zhang, N. Shah, P. Lopreiato, G. Todeschi, K. Ramakrishnan, and T. Wood, "OpenNetVM: A platform for high performance network service chains," Proc. 2016 Workshop on Hot Topics in Middleboxes and Network Function Virtualization, HotMIddlebox'16, pp.26–31, ACM, New York, NY, USA, 2016.
[13] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M.F. Kaashoek, "The Click modular router," ACM Trans. Comput. Syst., vol.18, no.3, pp.263–297, Aug. 2000.
[14] "The Click Modular Router Project." http://www.read.cs.ucla.edu/click/
[15] L. Rizzo, "Netmap: A novel framework for fast packet I/O," Proc. 2012 USENIX Conference on Annual Technical Conference, USENIX ATC'12, p.9, USENIX Association, Berkeley, CA, USA, 2012.
[16] "Cloud Networking Performance Lab." http://cnp.neclab.eu/clickos
[17] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," Proc. 6th Conference on Symposium on Opearting Systems Design & Implementation, OSDI'04, vol.6, p.10, USENIX Association, Berkeley, CA, USA, 2004.
[18] "Intel® Data Plane Development Kit (DPDK)." http://dpdk.org/
[19] "E2." http://span.cs.berkeley.edu/e2.html
[20] "PCI-SIG SR-IOV Primer: An Introduction to SR-IOV Technology." http://www.intel.com/content/www/us/en/pci-express/pci-sig-sr-iov-primer-sr-iov-technology-paper.html
[21] "openNetVM." http://sdnfv.github.io/onvm/
[22] B. Pfaff, J. Pettit, T. Koponen, E.J. Jackson, A. Zhou, J. Rajahalme, J. Gross, A. Wang, J. Stringer, P. Shelar, K. Amidon, and M. Casado, "The design and implementation of open vswitch," Proc. 12th USENIX Conference on Networked Systems Design and Implementation, NSDI'15, pp.117–130, USENIX Association, Berkeley, CA, USA, 2015.
[23] C. Wang, O. Spatscheck, V. Gopalakrishnan, Y. Xu, and D. Applegate, "Toward high-performance and scalable network functions virtualization," IEEE Internet Comput., vol.20, no.6, pp.10–20, Nov. 2016.
[24] M. Paolino, N. Nikolaev, J. Fanguede, and D. Raho, "SnabbSwitch user space virtual switch benchmark and performance optimization for NFV," 2015 IEEE Conference on Network Function Virtualization and Software Defined Network (NFV-SDN), pp.86–92, Nov. 2015.
[25] M. Dobrescu, N. Egi, K. Argyraki, B.G. Chun, K. Fall, G. Iannaccone, A. Knies, M. Manesh, and S. Ratnasamy, "RouteBricks: Exploiting parallelism to scale software routers," Proc. ACM SIGOPS 22nd Symposium on Operating Systems Principles, SOSP'09, pp.15–28, ACM, New York, NY, USA, 2009.
[26] S. Han, K. Jang, K. Park, and S. Moon, "PacketShader: A GPU-accelerated software router," SIGCOMM Comput. Commun. Rev., vol.40, no.4, pp.195–206, Aug. 2010.
[27] Y. Nakajima, T. Hibi, H. Takahashi, H. Masutani, K. Shimano, and M. Fukui, "Scalable, high-performance, elastic software OpenFlow switch in userspace for wide-area network." https://www.usenix.org/conference/ons2014/poster-session
[28] R. Rahimi, M. Veeraraghavan, Y. Nakajima, H. Takahashi, Y. Nakajima, S. Okamoto, and N. Yamanaka, "A high-performance OpenFlow software switch," 2016 IEEE 17th International Conference on High Performance Switching and Routing (HPSR), pp.93–99, June 2016.
[29] D. Zhou, B. Fan, H. Lim, D.G. Andersen, M. Kaminsky, M. Mitzenmacher, R. Wang, and A. Singh, "Scaling up clustered network appliances with ScaleBricks," SIGCOMM Comput. Commun. Rev., vol.45, no.4, pp.241–254, Aug. 2015.
[30] V. Srinivasan, S. Suri, and G. Varghese, "Packet classification using tuple space search," SIGCOMM Comput. Commun. Rev., vol.29, no.4, pp.135–146, Aug. 1999.
[31] "Open vSwitch." http://openvswitch.org
[32] D. Zhou, B. Fan, H. Lim, M. Kaminsky, and D.G. Andersen, "Scalable, high performance Ethernet forwarding with CuckooSwitch," Proc. Ninth ACM Conference on Emerging Networking Experiments and Technologies, CoNEXT'13, pp.97–108, ACM, New York, NY, USA, 2013.
[33] "cuckooswitch: A software-based Ethernet switch design built around a memory-efficient, high-performance, and highly-concurrent hash table for compact and fast FIB lookup." https://github.com/efficient/cuckooswitch
[34] R. Ierusalimschy, L.H. de Figueiredo, and W. Celes, "The evolution of Lua," Proc. Third ACM SIGPLAN Conference on History of Programming Languages, HOPL III, New York, NY, USA, pp.2-1–2-26, ACM, 2007.
[35] "Vfio kernel documentation." https://www.kernel.org/doc/Documentation/vfio.txt, May 2015.
[36] "Snabb: Simple and fast packet networking." https://github.com/snabbco/snabb
[37] "RouteBricks: Enabling General Purpose Network Infrastructure." http://routebricks.org/index.html
[38] "PacketShader - GPU-accelerated Software Router." https://shader.kaist.edu/packetshader/
[39] L. Rizzo, "The netmap project." http://info.iet.unipi.it/ luigi/netmap/
[40] "DPDK: Supported Operating Systems." http://dpdk.org/doc/guides/rel_notes/supported_os.html
[41] "DPDK: Supported NICs." http://dpdk.org/doc/nics

[42] J.D. Valois, "Implementing lock-free queues," Proc. Seventh International Conference on Parallel and Distributed Computing Systems, pp.64–69, Las Vegas, NV, 1994.
[43] "Intel® Data Direct I/O Technology." http://www.intel.com/content/www/us/en/io/data-direct-i-o-technology.html
[44] "Open vSwitch accelerated by Intel® DPDK." https://github.com/01org/dpdk-ovs
[45] A. Emani, "Using Open vSwitch with DPDK for Inter-VM NFV applications." https://software.intel.com/en-us/articles/using-open-vswitch-with-dpdk-for-inter-vm-nfv-applications, Dec. 2016.
[46] "Configure DPDK-accelerated Open vSwitch (OVS) for networking." https://access.redhat.com/documentation/en/red-hat-openstack-platform/10/paged/network-functions-virtualization-configuration-guide/chapter-3-configure-dpdk-accelerated-open-vswitch-ovs-for-networking
[47] G. Varghese, Network Algorithmics: An Interdisciplinary Approach to Designing Fast Networked Devices, Morgan Kaufmann Series in Networking, 2005.
[48] "Lagopus switch." http://www.lagopus.org
[49] J. Deng, H. Hu, H. Li, Z. Pan, K.C. Wang, G.J. Ahn, J. Bi, and Y. Park, "VNGuard: An NFV/SDN combination framework for provisioning and managing virtual firewalls," Proc. IEEE Conf. Network Function Virtualization and Software Defined Network (NFV-SDN), pp.107–114, Nov. 2015.
[50] "OPNFV." https://www.opnfv.org
[51] ESnet, "Firewall performance issues." https://fasterdata.es.net/network-tuning/firewall-performance-issues/
[52] M. Chapple, "Firewall rules are meant to be managed not broken." http://www.biztechmagazine.com/article/2012/08/firewall-rule-management-key-network-security, 2012.
[53] V.A. Olteanu, F. Huici, and C. Raiciu, "Lost in network address translation: Lessons from scaling the world's simplest middlebox," Proc. 2015 ACM SIGCOMM Workshop on Hot Topics in Middleboxes and Network Function Virtualization, HotMiddlebox'15, pp.19–24, ACM, New York, NY, USA, 2015.
[54] R. Mijumbi, J. Serrat, J. l. Gorricho, S. Latre, M. Charalambides, and D. Lopez, "Management and orchestration challenges in network functions virtualization," IEEE Commun. Mag., vol.54, no.1, pp.98–105, Jan. 2016.
[55] "ETSI GS NFV-IFA 009 V1.1.1: Network Functions Virtualisation (NFV); Management and Orchestration; Report on Architectural Options," July 2016.
[56] A. Mohammadkhan, G. Liu, W. Zhang, K.K. Ramakrishnan, and T. Woodv, "Protocols to support autonomy and control for NFV in software defined networks," 2015 IEEE Conference on Network Function Virtualization and Software Defined Network (NFV-SDN), pp.163–169, Nov. 2015.
[57] J.G. Herrera and J.F. Botero, "Resource allocation in NFV: A comprehensive survey," IEEE Trans. Netw. Serv. Manage., vol.13, no.3, pp.518–532, Sept. 2016.
[58] M. Xia, M. Shirazipour, Y. Zhang, H. Green, and A. Takacs, "Network function placement for NFV chaining in packet/optical datacenters," J. Lightwave Technol., vol.33, no.8, pp.1565–1570, April 2015.
[59] M. Xia, M. Shirazipour, Y. Zhang, H. Green, and A. Takacs, "Optical service chaining for network function virtualization," IEEE Commun. Mag., vol.53, no.4, pp.152–158, April 2015.
[60] F. Bari, S.R. Chowdhury, R. Ahmed, R. Boutaba, and O.C.M.B. Duarte, "Orchestrating virtualized network functions," IEEE Trans. Netw. Serv. Manage., vol.13, no.4, pp.725–739, Dec 2016.
[61] S. D'Oro, L. Galluccio, S. Palazzo, and G. Schembra, "Exploiting congestion games to achieve distributed service chaining in NFV networks," IEEE J. Sel. Areas. Commun., vol.35, no.2, pp.407–420, 2017.
[62] L. Qu, C. Assi, and K. Shaban, "Delay-aware scheduling and resource optimization with network function virtualization," IEEE Trans. Commun., vol.64, no.9, pp.3746–3758, Sept. 2016.
[63] P. Quinn and T. Nadeau, "Problem statement for service function chaining," RFC 7498 Informational, April 2015.
[64] J. Halpern and C. Pignataro, "Service function chaining (SFC) architecture." RFC 7665 (Informational), Oct. 2015.
[65] S. Ma, B. Wang, X. Zhang, and X. Gao, "ApplianceBricks: A scalable network appliance architecture for network functions virtualization," China Commun., vol.13, no.Supplement 1, pp.32–42, 2016.
[66] N. Egi, A. Greenhalgh, M. Handley, M. Hoerdt, F. Huici, and L. Mathy, "Towards high performance virtual routers on commodity hardware," Proc. 2008 ACM CoNEXT Conference, CoNEXT'08, pp.20:1–20:12, New York, NY, USA, ACM, 2008.
[67] M. Akagi, R. Usui, Y. Arakawa, S. Okamoto, and N. Yamanaka, "Cooperating superpeers based service-parts discovery for ubiquitous grid networking (uGrid)," 2008 7th International Conference on Optical Internet, pp.1–2, Oct. 2008.
[68] D. Ishii, K. Nakahara, S. Okamoto, and N. Yamanaka, "A novel IP routing/signaling based service provisioning concept for ubiquitous grid networking environment," 2010 IEEE Globecom Workshops, pp.1746–1750, Dec. 2010.
[69] S. Okamoto, N. Yamanaka, H. Takeshita, T. Okano, and T. Uchida, "Discussion on network virtualization," IEICE Technical Report on Network Systems, pp.433–438, March 2014 (in Japanese).
[70] Fujitsu White Paper, "Bringing Disaggregation to Transport Networks," Oct. 2015.
[71] M.D. Leenheer, T. Tofigh, and G. Parulkar, "Open and programmable metro networks," 2016 Optical Fiber Communications Conference and Exhibition (OFC), pp.1–3, March 2016.
[72] P. Gao, A. Narayan, S. Karandikar, J. Carreira, and S. Han, "Network requirement for resource disaggregation," 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI), pp.249–264, Nov. 2016.
[73] K. Kitayama, A. Hiramatsu, M. Fukui, T. Tsuritani, N. Yamanaka, S. Okamoto, M. Jinno, and M. Koga, "Photonic network vision 2020 – toward smart photonic cloud," J. Lightwave Technol., vol.32, no.16, pp.2760–2770, Aug. 2014.

Malathi Veeraraghavan received a B.Tech. degree from the Indian Institute of Technology (Madras) in 1984 and M.S. and Ph.D. degrees from Duke University in 1985 and 1988, respectively. She is a professor of the Charles L. Brown Department of Electrical and Computer Engineering at the University of Virginia. After a 10 year career at Bell Laboratories, she served on the faculty at Polytechnic University, Brooklyn, New York from 1999 to 2002. Her current research work on optical networks is supported by the US National Science Foundation (NSF) and the US Department of Energy (DOE). She holds 30 US patents and has received six Best Paper Awards. She served as the technical program committee chair for IEEE ICC 2002. She is a senior member of the IEEE.

Takehiro Sato received the B.E., M.E. and Ph.D. degrees in engineering from Keio University, Japan, in 2010, 2011 and 2016, respectively. He is currently a research associate in Graduate School of Science and Technology, Keio University, Japan. His research interests include communication protocols and network architectures for the next generation optical network. From 2011 to 2012, he was a research assistant in the Keio University Global COE Program, "High-level Global Cooperation for Leading-edge Platform on Access Spaces" by Ministry of Education, Culture, Sports, Science and Technology, Japan. From 2012 to 2015, he was a research fellow of Japan Society for the Promotion of Science. He is a member of the IEEE, and the IEICE.

Molly Buchanan received her B.A. in Computer Science from the University of Virginia in 2016. Among her interests are wireless computing and communication as well as computer science education.

Reza Rahimi is a second-year Ph.D. student in the Computer Engineering Program at the University of Virginia. He received his M.S. from Sharif University of Technology in the area of Computer Vision. His research interests are Cloud robotics and Software Defined Networking.

Satoru Okamoto received B.E., M.E. and Ph.D. degrees in electronics engineering from Hokkaido University, Hokkaido, Japan, in 1986, 1988 and 1994. He is currently a project Professor of Keio University, Japan. In 1988, he joined Nippon Telegraph and Telephone Corporation (NTT), Japan, where he conducted research on ATM cross-connect system architectures, photonic switching systems, and optical path network architectures and participated in the development of GMPLS-controlled HIKARI router ("photonic MPLS router") systems. He is an IEEE Senior Member.

Naoaki Yamanaka graduated from Keio University, Japan where he received B.E., M.E. and Ph.D. degrees in engineering in 1981, 1983 and 1991, respectively. In 1983 he joined Nippon Telegraph and Telephone Corporation's (NTT's) Communication Switching Laboratories, Tokyo Japan. He has been active in the development of ATM base backbone network and system including Tb/s electrical/optical backbone switching as NTT's Distinguished Technical Member. He moved to Keio University in 2014. He is currently a Professor in Dept. of Information and Computer Science, Vice Chair of Keio Leading-edge Laboratory of Science and Technology, and Chair of Photonic Internet Lab. He is an IEEE Fellow.