Re-architecting NFV Ecosystem with Microservices: State-of-the-art and Research Challenges

Shihabur Rahman Chowdhury, Mohammad A. Salahuddin, Noura Limam, and Raouf Boutaba
David R. Cheriton School of Computer Science, University of Waterloo, Waterloo, Canada
{sr2chowdhury | mohammad.salahuddin | n2limam | rboutaba}@uwaterloo.ca

***Abstract*—Network Functions Virtualization (NFV), considered a key enabler of Network "softwarization", promises to reduce the capital and operational expenditure for network operators by moving packet processing from purpose-built hardware to software running on commodity servers. However, the state-of-the-art in NFV is merely replacing monolithic hardware with monolithic Virtual Network Functions (VNFs), the software that realizes different network functions (*e.g.*, Firewalls, WAN Optimizers, *etc*.). Although this is a first step towards deploying NFV, common functionality is repeatedly implemented in monolithic VNFs. Repeated execution of such redundant functionality introduces processing overhead when VNFs are chained to realize Service Function Chains and leads to sub-optimal usage of infrastructure resources. This stresses the need for re-architecting the NFV ecosystem, from VNFs to their orchestration, through modular VNF design and flexible service composition. In that perspective, we make the case for using the *microservice* software architecture, proven to be effective for building large scale cloud applications from reusable and independently deployable components, to re-architect the NFV ecosystem. We also discuss the state-of-the-art in realizing modular VNFs from both industry and academia. Finally, we outline a set of research challenges for microservice-based NFV platforms.**

I. Introduction

Network operators ubiquitously deploy hardware *middleboxes* (*e.g.*, Firewalls, WAN Optimizers, *etc*.) to realize different security and performance goals and to offer value-added services (*e.g.*, parental control, video-on-demand, *etc*.). The number of middleboxes in a network is typically in the same order as the number of switches and routers [1]. However, the middlebox ecosystem is in many ways similar to the mainframe industry in the early '80s, *i.e.*, vendor specific, purpose-built and vertically integrated hardware, software and control with limited to no programmability. Such closed and inflexible ecosystem forces the network operators to resort to manual and error-prone methods for deploying and configuring network services. Deployment and configuration scenarios become further complicated when network operators need to steer traffic through an ordered sequence of middleboxes, *i.e.*, a Service Function Chain (SFC) (defined in IETF RFC7498). The inflexibilities in the middlebox ecosystem leads to increased operational complexity and expenditure, delaying time to market for new services.

More recently, the networking industry is going through a transformation known as *network softwarization*. Network softwarization refers to the general practice of decoupling network processing and control software from specialized hardware. This enables the network operators to replace vendor specific, purpose-built networking equipment with commodity hardware and leverages open APIs to control and provision the networking infrastructure in a programmatic way. A key enabler for network softwarization is *Network Functions Virtualization (NFV)* [2]. NFV proposes to virtualize the Network Functions (NFs), *i.e.*, functionality realized by the hardware middleboxes, by decoupling the traffic processing software from purpose-built hardware. In this way, NFV enables the operators to replace vendor specific, expensive and vertically integrated hardware middleboxes with *Virtual Network Functions (VNFs)* running on commodity off-the-shelf servers. This increases flexibility and re-usability of the same hardware for multiple NFs. Hence, NFV brings economies of scale and leverages the advances in application orchestration for on-demand deployment and scaling of SFCs.

NFV is envisioned to increase the agility of the communication infrastructure to support future applications, *e.g.*, IoT, smart grid, smart cities, connected drones, *etc*. However, the state-of-the-art NFV platforms (*e.g.*, OPNFV (https://www.opnfv.org/), OpenMANO (https://osm.etsi.org/) are merely replacing monolithic hardware NFs with their monolithic software counterparts. Definitely, this is the first logical step towards network softwarization transformation. However, monolithic VNFs in SFCs can lead to sub-optimal resource usage and hinder infrastructure agility.

A fundamental problem with monolithic VNFs is that a large number of common functionalities (*e.g.*, packet header parsing, payload inspection, packet classification, *etc*.) are repeated across different VNFs developed by various third-party providers. This has several negative consequences, including: (i) redundant development of functionalities in different VNFs; (ii) unnecessary processing overhead for executing these redundant functionalities when VNFs are chained to form SFCs (shown to exceed 25% for some SFCs [3]); and (iii) coarse-grain resource allocation and scaling imposed by the monolithic nature of VNFs. These limitations stresses the need to rethink how VNFs can be developed and orchestrated for agile service creation and scaling [4].

In this article, we make the case for re-architecting the current monolithic VNFs and their orchestration to eliminate inherent redundancies and exploit their commonalities through modular VNF design and flexible service composition. To this end, we envision the adoption of the *microservice software architecture* [5] for building, deploying and managing VNFs. The microservice software architecture decomposes a monolithic software into independently deployable components

with well-defined interfaces, called *microservices*. It has been proven effective for building large-scale cloud applications. In the remainder of this article, we start by providing background on the microservice software architecture. Then we make our case for using microservices to re-architect VNFs through an illustrative example. This is followed by a discussion on recent efforts from industry and academia towards the development of modular VNFs and their supporting runtime systems. Finally, we outline several research challenges that we have identified for realizing microservice-based VNFs.

II. Microservice Software Architecture

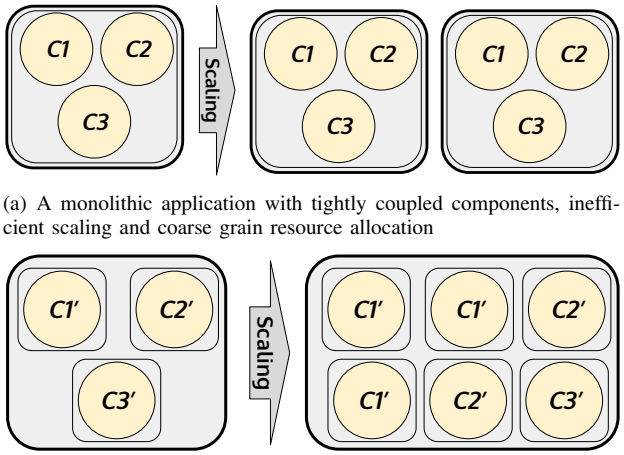
The National Institute of Standards and Technology (NIST) defines microservices as follows. *"A microservice is a basic element that results from the architectural decomposition of an application's components into loosely coupled patterns consisting of self-contained services that communicate with each other using a standard communications protocol and a set of well-defined APIs, independent of any vendor, product or technology"* [5]. From this definition, we draw the following key features of microservices:

- **Self-contained:** A microservice typically performs a single task and does not depend on other microservices for performing its task.
- **Loosely coupled:** Generally, a microservice does not require tight synchronization with other microservices to operate.
- **Well defined communication interfaces:** In a microservice architecture, an application completes a series of tasks by different microservices. For this purpose, microservices must exhibit well-defined APIs that an orchestrator and other microservices can use for communication. A popular choice for such interface is the RESTful API. However, message queues and protocol buffers are competing alternatives.

A major advantage of using microservices is the ability to allocate resources at a finer granularity. For example, consider a monolithic application in Fig. 1(a), composed of tightly coupled components, $C1$, $C2$ and $C3$. A surge in demand requires increased processing in components $C1$ and $C2$ of the application. In this monolithic application, all components need to be scaled because of the tight coupling, resulting in idle resources allocated to $C3$. In contrast, consider Fig. 1(b), where the same application is re-architected using microservices $C1'$, $C2'$ and $C3'$. Now the scaling of the application is more efficient in terms of resource allocation. Additionally, the self-contained and loosely coupled nature of microservices facilitate independent development and maintenance.

III. The Case for Adopting Microservices for NFV

In this section, we motivate the adoption of microservice software design principle for re-architecting VNFs. We use the example SFC in Fig. 2(a), which is described in an Internet draft for SFC use cases in data center networks (https://tools.ietf.org/html/draft-ietf-sfc-dc-use-cases-06), to illustrate the common functionalities between different VNFs. The details of each function in Fig. 2(a) can be found in the



(a) A monolithic application with tightly coupled components, inefficient scaling and coarse grain resource allocation

(b) A microservice-based application with independently scalable microservices, facilitating fine grain resource allocation

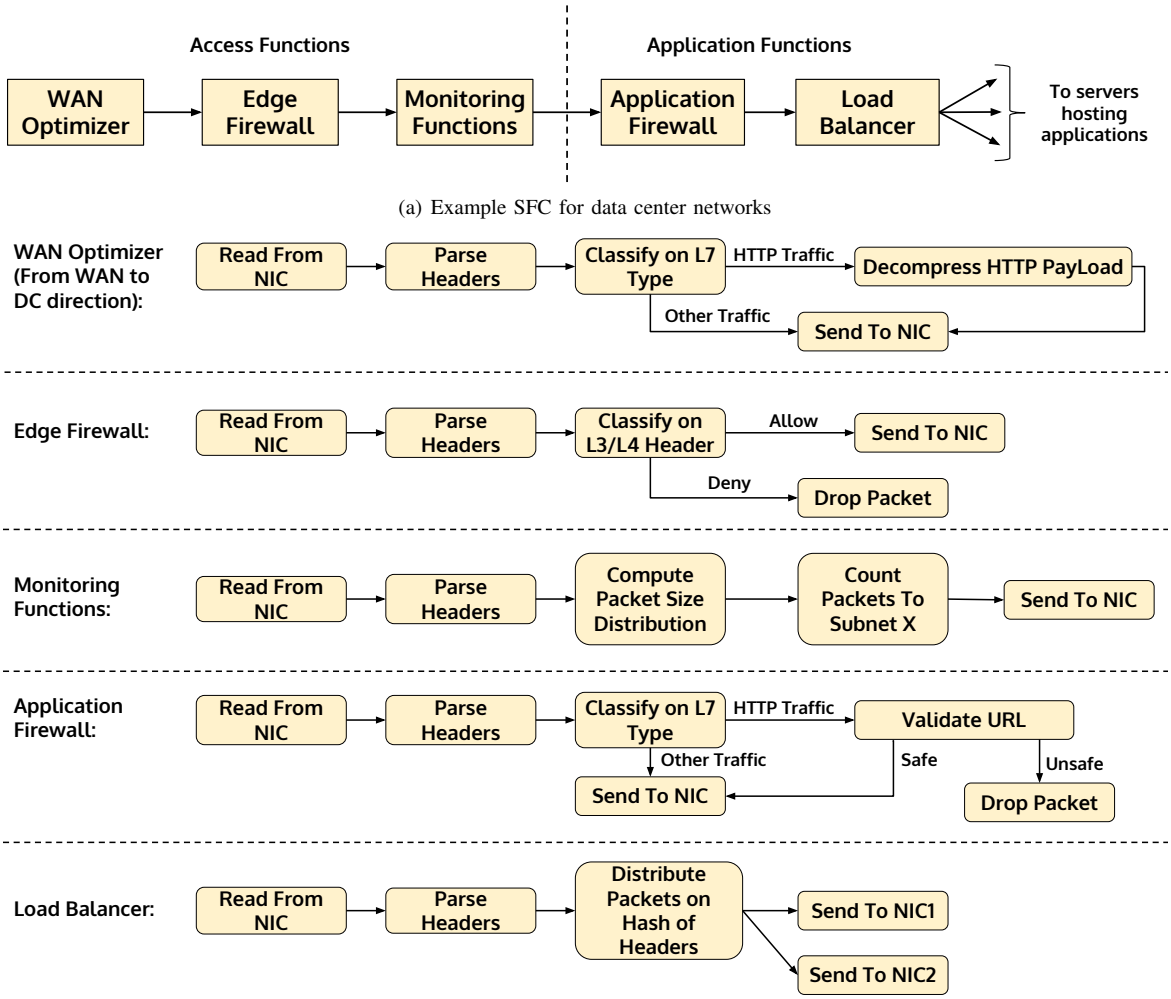
Fig. 1: Monolithic vs. microservice-based application

Internet draft. In the following, we briefly describe the specific functionality for our subsequent discussion:

- **WAN Optimizer:** compresses/decompresses HTTP payload (*e.g.*, text, image) on egress/ingress traffic to optimize bandwidth usage on the WAN links that connect a data center to another peer data center.
- **Edge Firewall:** performs access control based on layer 2–4 headers, *e.g.*, IP subnet, TCP port, *etc*.
- **Monitoring Function:** performs the following operations: (i) compute the packet size distribution from network traffic; and (ii) count the number of packets destined to an IP subnet inside the data center network.
- **Application Firewall:** blocks incoming HTTP requests that have an SQL injection attack embedded in the URL.
- **Load Balancer:** distributes traffic to the back-end servers based on the hash of layer 3–4 headers.

We perform a functional decomposition of each of the aforementioned VNFs and present it in Fig. 2(b). Each block in Fig. 2(b) represents an operation performed on a packet while the arrows represent the flow of operations. We observe the following from the decomposition:

- **Overlapped Functionality:** Many of the functionality are common across VNFs. In some cases, the functionality is the same but the parameters are different (*e.g.*, writing a packet but to different Network Interface Cards (NICs)). While in other cases both the functionality and the configuration are exactly the same (*e.g.*, HTTP packet classification performed by both WAN Optimizer and Application Firewall).
- **Wasted CPU Cycles:** Since a functionality is embedded within a monolithic VNF, its execution result is not easily reusable across VNFs (*e.g.*, classification of a packet as HTTP by the WAN Optimizer is not usable by the Application Firewall). Therefore, CPU cycles are wasted due to the repeated execution of the same functionality when a packet goes through the VNFs in an SFC. This is also experimentally validated by prior research, reporting



(a) Example SFC for data center networks

(b) Functional decomposition of the VNFs in Fig. 2(a)

Fig. 2: Illustrative example to motivate microservice-based VNFs

more than 25% CPU cycles are wasted for such redundant processing for certain SFCs [3].

- **Inflexible Scaling:** It is difficult to scale resources for individual functionality since they are embedded into monolithic VNFs. If a functional block in Fig. 2(b) becomes a bottleneck, the whole VNF needs to be scaled up/out (*e.g.*, there is no easy way to allocate additional CPU resources for performing URL validation in an Application Firewall in Fig. 2(b) without scaling up/out the whole VNF instance). This requires more resources compared to scaling up/out resources for a single functionality.

These observations motivate the adoption of microservices for re-architecting the VNFs. Applying the microservice software design principle to VNFs will result in a set of independently deployable packet processing microservices with well-defined interfaces corresponding to the functional blocks in Fig. 2(b). In what follows, we use the term μNF to refer to these packet processing microservices. An SFC will then be realized by orchestrating a processing graph composed of μNFs while removing repeated application of the same packet processing functionality. As a result, it will be possible to recover the CPU cycles spent in redundant processing while performing finer grain resource allocation.

μNFs are to some extent analogous to the VNF Components (VNFCs) described in ETSI's VNF architectural description [6]. According to [6], a VNF consists of one or more VNFCs, where each VNFC has a 1:1 correspondence with a virtualization container (*e.g.*, VM, OS container, *etc*.). However, a major difference between a μNF and a VNFC is that μNFs are loosely coupled components that can be used across VNF boundaries, whereas VNFCs are not. This property also makes SFC orchestration different in the case of μNFs since there are more optimization opportunities for μNFs compared to VNFCs.

In Fig. 3, we present an optimized realization of the SFC from Fig. 2(a) using μNFs to illustrate some of the optimization opportunities. Assuming that mechanisms to propagate output of one μNF to the others exist, we can consolidate packet processing functions such as packet I/O, header parsing, HTTP packet classification, *etc*. Furthermore, some μNFs can be executed in parallel, such as the counting functionality

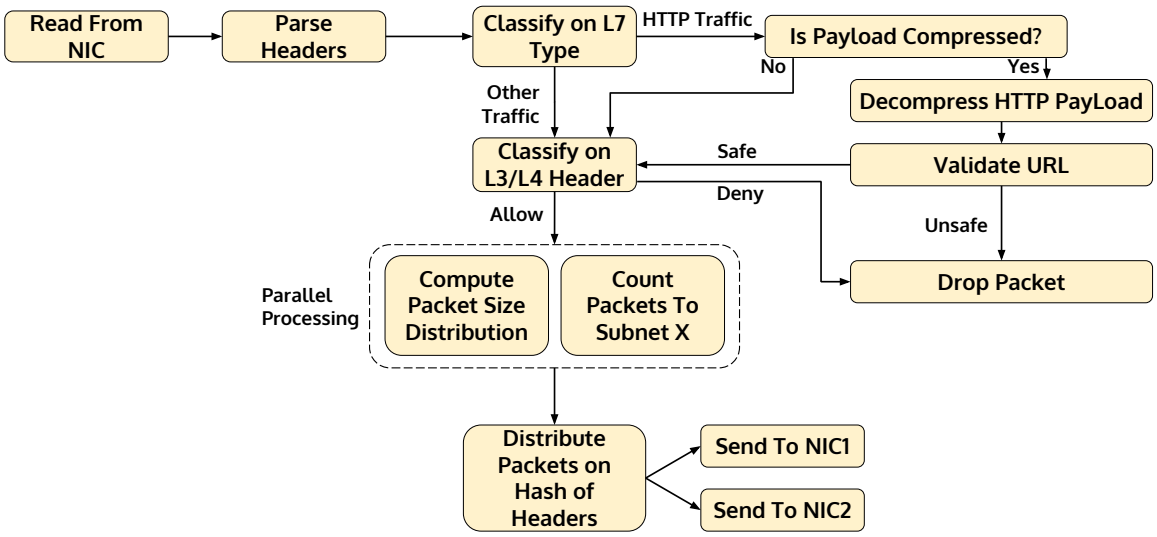


Fig. 3: Microservice-based realization of the SFC in Fig. 2(a)

performed by the monitoring functions that do not modify packets. These optimizations are only possible due to the inherent loosely coupled characteristic of μNFs.

IV. State-of-the-Art

The history of modular packet processing using software dates back to the late '90s with the introduction of *Click* [7] modular router. However, Click is not built based on the principles of microservices. It compiles a monolithic packet processing software from a set of reusable packet processing elements. The concept of modularization in Click motivated later efforts to address the shortcomings of monolithic VNFs. Although, these endeavors did not always explicitly label their approach as using microservices, they follow the principles of a microservice architecture. In this section, we briefly discuss the efforts from both academia and industry that address the shortcomings of monolithic VNFs through a modular design.

A. Academic Efforts

CoMb [3] is one of the early endeavors that partially addressed the short-comings of monolithic VNFs. CoMb proposed to share some low-level common functionality between VNFs (*e.g.*, TCP session reconstruction). However, apart from a set of low-level common functionalities, the VNFs in CoMb were monolithic. Some recent efforts in the literature have proposed to replace monolithic VNFs with reusable data path components that can be controlled and configured from a centralized control plane. While others have developed runtime systems to support VNFs composed from these data plane processing functions. In the following, we discuss the state-of-the-art academic efforts and categorize them based on architectures for μNF-based VNFs and runtime systems to support μNF-based VNFs, respectively.

1) μNFs-based VNF Architectures: OpenBox [8] and Microboxes [9] are two recent endeavors that address the short-comings of monolithic VNFs. Both propose an architecture

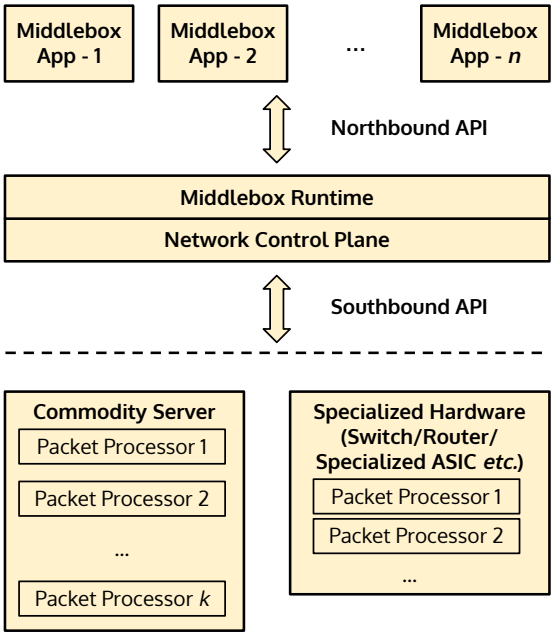


Fig. 4: Generalized architecture of OpenBox [8] and Microboxes [9]

for composing VNFs from lightweight and reusable packet processing components. Their proposed architectures have similarities and can be generalized as depicted in Fig. 4. They realize VNFs and SFCs using a composition of lightweight packet processing modules (*OpenBox Service Instance (OBSI)* in OpenBox and *μStack* in Microboxes) running on commodity servers or possibly on specialized hardware. OBSIs and μStacks are analogous to μNFs discussed in Section III. An application written in a high-level language (*e.g.*, Python) determines the composition of OBSIs or μStacks (*cf.* Middlebox Apps in Fig. 4). Necessary abstractions for writing applications are provided by a middlebox runtime system (*OpenBox control*

plane or *Microboxes controller*). The runtime system is also responsible for orchestration tasks such as placing OBSIs or μStacks and steering traffic between them.

A fundamental difference between OBSIs and μStacks is the way they are executed in a packet processing pipeline. OBSIs are executed sequentially and interact with each other using a packet centric communication model, *i.e.*, a batch of packets is sequentially processed by the OBSIs in a packet processing pipeline until the batch exits the system. In contrast, μStacks are executed asynchronously and in parallel, and communicate with each other using an event based publish/subscribe model. A μStack publishes TCP processing events (*e.g.*, connection initiation or termination) and the associated meta-data (*e.g.*, pointer to SYN or FIN packets) to other μStacks that have subscribed to these events. The subscribed μStacks are then executed, possibly in parallel on the packets from the same flow. The sequential execution of OBSIs in OpenBox is simpler and easier to reason when compared to the asynchronous and parallel execution of μStacks in Microboxes. However, the latter reduces packet processing latency and better utilizes the CPUs of the underlying machines.

2) Runtime Systems for μNF-based VNFs and SFCs: Flurries [10] and NetBricks [11] address issues related to the runtime system for supporting VNFs composed of components built around the concept of μNFs. However, they focus on different aspects of the runtime system. Flurries proposes an efficient packet exchange mechanism between μNFs running on Docker containers. Whereas, NetBricks proposes isolation mechanisms between multiple instances of μNF-like components without using VMs or OS containers.

Flurries assumes the existence of thousands of lightweight NFs (similar to μNFs) running in Docker containers, each performing a small packet processing task while processing packets from a single flow. Under these assumptions, Flurries proposes the design of a data plane for the NFs. The proposed data plane provides zero-copy packet exchange mechanism between the NF containers deployed on the same machine. It performs flow classification and coordinates with a scheduler to assign new flows to new NFs. A zero-copy packet exchange mechanism is provided by using the shared memory primitives from Intel DPDK library. The NFs running in containers use a hybrid of interrupt and polling techniques to maximize throughput while not always using 100% of the CPU. However, the concept of ownership transfer as provided by virtual switches (*e.g.*, Open vSwitch) is not implemented in Flurries. Ownership transfer ensures that an NF, which has finished processing a packet should not be able to use the memory handlers to modify the same packet in the future.

The authors of NetBricks argue through an experimental study that current virtualization technologies (*e.g.*, VMs, OS containers) cannot meet the performance requirement of NFV. According to their study, packet processing performance for 64 byte packets drops by $3\times$ and $7\times$ for containers and VMs, respectively, when compared to bare-metal performance. NetBricks addresses these limitations by designing a process-based runtime system, which provides the same memory isolation as containers and VMs without incurring the performance overhead. It also provides packet processing abstractions to compose VNFs. These abstractions are similar to the concept of μNFs, *i.e.*, they also separate the packet processing and the pipeline composition.

However, NetBricks compiles everything into one NetBricks process and runs each SFC in a separate thread inside that process. Within one SFC, *i.e.*, within one thread in NetBricks, a NF passes its processed packets to the next NF by invoking the next NF's appropriate method, which in turn ensures the transfer of packet ownership. Furthermore, NetBricks relies on Rust, a memory safe programming language that prevents unsafe memory operations performed by multiple threads. Although, NetBricks does not strictly adhere to the concept of μNFs, the proposed runtime system can be a potential choice for deploying μNFs.

Flurries and NetBricks address specific issues regarding the runtime system, *i.e.*, packet exchange mechanism and memory isolation. However, they do not address issues such as μNFs placement, fault-tolerance, state management, *etc*. They rely on external systems to provide such functionalities.

B. Comparison between Academic Approaches

Table I presents a comparative study of the academic efforts based on the following features:

- **Abstraction for processing:** This refers to the level of abstraction exposed by the underlying runtime system for developing μNFs (*e.g.*, packet, flow, event). Also, a μNF can have "one-to-one" or "one-to-many" relationship with the processing granularity, *e.g.*, one μNF per flow (one-to-one relationship between μNF and flow), one μNF for a set of flows (one-to-many relationship between μNF and flow).
- **Placement and resource allocation:** This relates to the process of determining the placement and the number of μNFs, and allocating sufficient resources to meet the desired QoS requirements of the SFC. We classify the proposals based on whether they integrate such mechanisms or rely on an external system.
- **Fault-tolerance:** This pertains to the necessary steps to ensure high availability of the SFCs.
- **Inter μNF communication:** The mechanism used by μNFs to exchange packets or events between them. This can be based on shared memory between processes, through a virtual switch, through method invocation, *etc*.
- **Memory isolation mechanism:** This refers to the mechanisms adopted to provide memory isolation between μNFs. Possible options include using VMs or OS containers, resorting to memory safe programming language and runtime system (*e.g.*, Rust and LLVM).
- **Ownership transfer:** This ensures that once a μNF has sent a packet to another μNF, it cannot modify the packet using the previously used packet handlers.

The above set of features, though not exhaustive, constitutes the collective feature set inspired from the state-of-the-art. The comparison based on these features provides an insight into the lack of maturity of this area of research and highlights the opportunities for further research. For example, we can see that the ownership transfer between multiple OS containers without copying packets is an open issue.

TABLE I: Comparison of the Academic State-of-the-art

State-of-the-art	Feature					
	Abstraction for Processing	**Placement & Resource Allocation**	**Fault Tolerance**	**Inter μNF Communication**	**Memory Isolation Mechanism**	**Ownership Transfer**
Microboxes [9]	Event (one-to-many)	N.A.	N.A.	Zero-copy Shared memory	N.A.	Not transferred
OpenBox [8]	Flow (one-to-many)	Heuristic	N.A.	N.A.	VM	N.A.
Flurries [10]	Flow (one-to-one)	External	N.A.	Zero-copy Shared memory	Docker container	Not transferred
NetBricks [11]	Packet, Bytestream (one-to-many)	External	External	Method invocation inside same address space	Threads inside processes built using Rust's memory safe operations	Inter-NF procedure call marks packets to ensure ownership transfer

N.A. = Not Addressed

C. Industry Efforts

There has been some effort from the industry on re-architecting VNFs using microservices. The Central Office Re-architected as Datacenter (CORD) project, led by Open Networking Lab (ON.Lab), focuses on transforming operator central offices by replacing the purpose-built hardware with commodity hardware. CORD also virtualizes a number of middleboxes such as Optical Line Terminator, Broadband Network Gateway, *etc*. to their virtual counterparts. A design approach taken during the process was to decouple the control and processing functions of these middleboxes and run them in separate VMs. As a result, scaling out data processing functionality does not incur the overhead of scaling redundant control plane functionalities. This decoupling can be considered as a step towards microservice decomposition for NFs.

Another industry effort to leverage the benefits of microservices in telecommunication networks is from Metaswitch Networks. Clearwater IMS[1] is their recent IP Multimedia Subsystem (IMS), which has been developed using the microservices design principle. Clearwater IMS can be deployed as a collection of independent software components, where each component can be horizontally and independently scaled. Clearwater stores the sate of each component in persistent storage, thus eliminating the complexity of state management while scaling the components. However, the components themselves are large monolithic software components including a substantial number of tightly coupled functionalities.

A fundamental difference observed between the industry and academic efforts is the granularity of microservice decomposition. The industry efforts have favored coarse-grain decomposition, compared to the fine-grain decomposition at the level of packet operations, promoted by academic projects.

V. Research Challenges

Designing, deploying and managing microservice-based software comes with its own set of challenges. Microservice-based NFV platforms inherit these challenges and add new dimensions due to inherent differences between VNFs and cloud applications. These differences include several orders of magnitude tighter latency overhead budget for VNFs compared to cloud applications, and higher reliability requirements (five nines) for network operators than cloud service providers (four or four and a half nines). In this section, we discuss the research challenges confronting the realization of microservice-based NFV ecosystem. Interested readers are directed to [12] and [13] for issues specific to microservices and NFV, respectively.

[1]http://www.projectclearwater.org/technical/clearwater-architecture/

A. Microservices decomposition of VNFs

The first challenge in realizing any microservice-based software is to identify the set of microservices. This is difficult because it requires domain specific knowledge. In case of NFV, we need to identify and develop a set of sufficiently generic μNFs that are able to realize a wide range of VNFs. One way to approach VNF decomposition is to leverage domain expertise (*e.g.*, by consulting with VNF developers) or to study existing open-source VNFs and identify smaller functional units. A μNF may perform one or more low-level protocol processing tasks corresponding to each layer in the TCP/IP stack. For example, classify packets on layer-2/3 header, re-construct a TCP stream, *etc*., or application-layer specific processing such as compressing or decompressing text/image in HTTP payload. Indeed, a key research issue here is to determine the granularity of such μNF operations, *i.e.*, single packet processing operation vs. a collection of operations performing a single functionality. On one hand, recent developments in modular VNF design propose low-level packet processing modules as the VNF building blocks ([8], [9]). On the other hand, state-of-the-art commercial VNFs (*e.g.*, Clearwater IMS) are being constructed from coarse-grain microservices. There are advantages and disadvantages of both approaches. A fine-grain decomposition facilitates better re-usability at the cost of increased communication overhead and higher packet processing latencies. In turn, a coarse-grain decomposition reduces re-usability but has the potential to reduce the aforementioned overheads. Therefore, it will be very useful to theoretically analyze and empirically evaluate this trade-off to determine the granularity that is most beneficial for VNF decomposition.

B. Communication Primitives for VNF Composition

Microservice software architecture mandates that each microservice has a well-defined interface for communicating with other microservices. Traditionally, cloud applications built using microservices have been using RESTful APIs or

message broker for communication, which will incur significant overhead when used for VNF composition. In case of composing VNFs from μNFs, the communication interface should have the following properties: (i) minimal impact (order of few microseconds) on the overall packet processing latency, (ii) ensure packet ownership transfer between μNFs, (iii) allow microservices to pass the result of their processing to other microservices to eliminate redundant processing, and (iv) enable μNF — μNF communication across physical machine boundary. Two design extremes of such communication interface are to perform a packet copy between microservices and to share packets using a shared memory, respectively. The first approach has significant performance implications albeit providing isolation, whereas the latter will incur low latency at the cost of isolation.

C. Resource Allocation for QoS-aware VNF Composition

The ability to use μNFs beyond VNF boundaries creates opportunities for optimizing resources allocated to SFCs. Recall from Section III, there are possibilities of consolidating multiple μNFs and also parallelize the execution of a subset of μNFs in an SFC. A systematic approach is required to develop algorithms for resource allocation for μNFs that take the aforementioned possibilities into account while meeting QoS requirements (*e.g.*, throughput). Other possibilities to consider include reordering the μNFs without changing the SFC semantics to improve existing placements or make new ones feasible, which were not possible with monolithic VNFs. Also, algorithms for dynamically scaling microservices to meet change in traffic demand also needs attention from the research community.

D. Microservice Performance Profiling

A key to efficient resource allocation and VNF management is to have accurate resource usage and performance profiles of the μNFs. The performance profile of μNFs will contain resources required (*e.g.*, CPU share) to achieve certain QoS parameters (*e.g.*, packets processed per second) under a given workload. Such profiles help the orchestration system to optimize μNF placement and the operators to reason about the perceived performance. This non-trivial task requires developing workloads that represent realistic scenarios and instrumentation mechanisms to measure different performance parameters under various loads.

E. Overhead vs. Flexibility

From an architectural perspective, microservices are an attractive choice for re-designing network functions. However, from a practical standpoint such redesign can also add overhead to packet processing (*e.g.*, increased processing latency due to long paths in μNF processing graph). A systematic study is required to understand the trade-offs between the observed overhead and the achieved flexibility. Such a study will help to identify the use cases where microservice-based NFs are more beneficial.

F. Language for VNF and SFC Composition

A fundamental requirement in a microservice-based NFV platform is a language that can capture the hierarchical nature of SFC composition, *i.e.*, how VNFs are composed using μNFs (a VNF template) and subsequently how SFCs will be composed from VNFs (SFC specification). When a network operator defines an SFC by describing the VNFs and their interconnections, the first step for the orchestrator is to translate the SFC or a set of SFCs to a μNF processing graph by leveraging a set of predefined VNF templates. This translation acts as a basis for further optimizing the μNF graph for resource allocation. Available orchestration languages such as TOSCA[2] and HOT[3] can be evaluated for possible adoption or new languages need to be developed.

G. Fault Tolerance and Recovery

In general, fault tolerance has been a key concern for microservice-based software. Re-usability, a key feature of microservices is also responsible for causing cascading failures. However, the situation becomes even more critical for telecommunication network operators who typically require higher reliability (five nines or more) compared to cloud service providers (four nines). A major challenge in realizing microservice-based NFV platforms is to ensure strong consistency of μNF states while recovering from failures within order of milliseconds. And all this needs to be achieved with an overhead budget of a few microseconds per packet.

H. Stateless vs. Stateful Microservice Design

A design choice in building microservice-based VNFs is to determine the placement of states (*e.g.*, address translation table). One approach is to keep the states with the microservices for easier maintenance. However, this makes scaling difficult due to state synchronization. Another approach is to maintain VNF specific state external to the microservices and have a separate logic for handling it (*e.g.*, similar to [14]). This decouples the microservices from stateful network processing and facilitates easier scaling. However, this can add to the packet processing latency due to remote state access. Therefore, the challenge lies in striking a balance between ease of scaling and maintenance overhead in such a platform.

I. Virtualization Platforms

Virtualization provides isolation between coexisting applications. However, the level of virtualization has an impact on the level of isolation and the performance overhead. A popular choice for deploying microservices in the cloud is containers (*e.g.*, Docker). However, studies show severe performance degradation in packet processing throughput even when using containers [11]. Ideally, the virtualization platform should have minimal performance impact compared to bare-metal and should be able to spawn μNFs within milliseconds to support on-demand service provisioning. This mandates the need for

[2]https://docs.oasis-open.org/tosca/tosca-nfv/v1.0/tosca-nfv-v1.0.html
[3]http://docs.openstack.org/developer/heat/template_guide/hot_guide.html

new virtualization platforms to support μNFs. One research direction is to explore *unikernel* operating systems [15] to create lightweight virtualization platform for μNFs.

J. Underlying Computer Architectures

Appropriate computer architectures to support μNF-based VNFs is an important consideration. State-of-the-art multi-core processors have tens of processing cores, reaching up to hundreds with multiple CPU sockets. However, inter-core communication and synchronization is considered expensive, let alone its inter-socket counterpart. A possible area of exploration is to use processors with hundreds of small and power-efficient cores to allow many microservices to be co-located on a single machine. In this regard, rack-scale computing architectures, *i.e.*, servers with a large number of tightly integrated system-on-a-chip, is a possible candidate for exploration.

K. Monitoring

Monitoring is an integral part of any network management system. A key challenge in microservice-based NFV platforms will be to derive end-to-end metrics (*e.g.*, end-to-end latency) from μNF specific monitoring data (*e.g.*, per packet latency). The challenges come from the fact that packets can take different paths in the μNF processing graph. Therefore, tracing end-to-end packet life-cycle will require understanding the interaction between microservices. This will require developing the necessary infrastructure to ingest μNF – μNF interaction and μNF specific metrics, and to analyze the data to obtain end-to-end performance metrics. Another design consideration is to equip the microservices with necessary monitoring hooks (*e.g.*, counters, filters *etc*.) in the first place to facilitate the derivation of end-to-end performance metrics.

L. Debugging and Verification

Debugging and verification are essential tools for both the developers and the network operators. This has been a well-investigated research topic for different kinds of networks. An additional challenge in μNF based deployment will be the scale of the problem. A μNFs-based deployment will increase the scale of NFV deployment by at least an order of magnitude. In addition, the debugging and verification tools must be able to capture the interaction between different μNFs. Capturing the interaction between μNFs, analyzing the interaction to capture the semantics of SFCs, verification of the semantics against network policies and identifying root cause of errors are some interesting research problems to explore.

M. Co-existence with traditional NFV Ecosystem

The transformation from monolithic VNFs to μNF-based ones will not happen overnight. Rather, we envision the two ecosystems to co-exist for some time. This poses challenges in designing the orchestrator for instance. In a hybrid deployment, the orchestrator should be able to communicate with both monolithic VNFs and μNFs. There is also room for alternative deployment models where each ecosystem has its own local orchestrator, managed by a global orchestrator.

VI. Conclusion

In this article, we motivate the adoption of the microservice architecture for re-architecting the NFV ecosystem. The ultimate goal is to speed up innovation in NFV by: (i) allowing independent development of small pieces of the ecosystem; (ii) improve the utilization of underlying infrastructure thus positively impacting operational costs; and (iii) contribute to increasing the infrastructure agility for supporting the next generation of applications. We also surveyed the state-of-the-art from both industry and academia. Our survey indicates that research in this field is still in its infancy and provides great opportunities for future contributions. The research challenges presented in this article can constitute a starting point for the interested readers in this exciting area.

Acknowledgement

This work was supported in part by the NSERC CREATE for Network Softwarization program.

References

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

[2] "Network Functions Virtualisation Introductory White Paper," White paper, Oct 2012, accessed: Feb 05, 2017. [Online]. Available: https://portal.etsi.org/nfv/nfv_white_paper.pdf

[3] V. Sekar, N. Egi, S. Ratnasamy, M. K. Reiter, and G. Shi, "Design and implementation of a consolidated middlebox architecture," in *Proc. of USENIX NSDI '12*. USENIX Association, 2012, pp. 24–24.

[4] R. Roseboro, "Cloud-native nfv architecture for agile service creation & scaling," White paper, Jan 2016.

[5] "NIST Definition of Microservices, Application Containers and System Virtual Machines," draft, Feb 2016, accessed: Feb 05, 2017. [Online]. Available: http://csrc.nist.gov/publications/drafts/800-180/sp800-180_draft.pdf

[6] "Network Functions Virtualisation (NFV); Virtual Network Functions Architecture," White paper, Dec 2014, accessed: Feb 05, 2017. [Online]. Available: http://www.etsi.org/deliver/etsi_gs/NFV-SWA/001_099/001/01.01.01_60/gs_NFV-SWA001v010101p.pdf

[7] R. Morris, E. Kohler, J. Jannotti, and M. F. Kaashoek, "The click modular router," in *Proc. of ACM SOSP '99*. ACM, 1999, pp. 217–231.

[8] A. Bremler-Barr, Y. Harchol, and D. Hay, "Openbox: a software-defined framework for developing, deploying, and managing network functions," in *Proc. of ACM SIGCOMM '16*. ACM, 2016, pp. 511–524.

[9] G. Liu, Y. Ren, M. Yurchenko, K. Ramakrishnan, and T. Wood, "Microboxes: high performance nfv with customizable, asynchronous tcp stacks and dynamic subscriptions," in *Proc. of ACM SIGCOMM 2018*. ACM, 2018, pp. 504–517.

[10] W. Zhang, J. Hwang, S. Rajagopalan, K. Ramakrishnan, and T. Wood, "Flurries: Countless fine-grained nfs for flexible per-flow customization," in *Proc. of ACM CoNEXT '16*. ACM, 2016, pp. 3–17.

[11] A. Panda, S. Han, K. Jang, M. Walls, S. Ratnasamy, and S. Shenker, "Netbricks: Taking the v out of nfv," in *Proc. of USENIX OSDI '16*. USENIX, 2016.

[12] N. Dragoni, S. Giallorenzo, A. L. Lafuente, M. Mazzara, F. Montesi, R. Mustafin, and L. Safina, "Microservices: yesterday, today, and tomorrow," in *Present and Ulterior Software Engineering*. Springer, 2017, pp. 195–216.

[13] R. Mijumbi, J. Serrat, J.-L. Gorricho, N. Bouten, F. De Turck, and R. Boutaba, "Network function virtualization: State-of-the-art and research challenges," *IEEE Comm. Surveys & Tutorials*, vol. 18, no. 1, pp. 236–262, 2016.

[14] M. Kablan, A. Alsudais, E. Keller, and F. Le, "Stateless network functions: Breaking the tight coupling of state and processing." in *Proc. of USENIX NSDI*, 2017, pp. 97–112.

[15] A. Madhavapeddy, R. Mortier, C. Rotsos, D. Scott, B. Singh, T. Gazagnaire, S. Smith, S. Hand, and J. Crowcroft, "Unikernels: Library operating systems for the cloud," in *ACM SIGPLAN Notices*, vol. 48, no. 4. ACM, 2013, pp. 461–472.