Journal of Internet Services and Applications, 2022, 12:1, doi: 10.573/jisa.2022.2342

NFV-COIN: Unleashing The Power of In-Network Computing with Virtualization Technologies

Giovanni Venâncio [**Federal University of Paraná** | ***gvsouza@inf.ufpr.br***]
Rogério C. Turchetti [**Federal University of Santa Maria** | ***turchetti@redes.ufsm.br***]
Elias P. Duarte Jr. [**Federal University of Paraná** | ***elias@inf.ufpr.br***]

Received 18 November 2021 • **Accepted** 26 July 2022 • **Published** 18 November 2022

Abstract
Network Functions Virtualization (NFV) allows the implementation in software of middleboxes traditionally available as specialized hardware. Network services can be implemented as SFCs (Service Function Chains) based on virtualization technologies that run on commodity hardware. Although most virtualized functions have classic middlebox functionalities (e.g. firewalls or intrusion detectors) NFV technology can be used to leverage the network to provide novel types of services to end-users. Actually, NFV can be very convenient to deploy traditional end-user services in the network, in the paradigm that has been called Computing In the Network (COIN). This article discusses the requirements to deploy COIN services using NFV technologies, which we call NFV-COIN. We also present case studies and an NFV-COIN architecture that is compliant with the NFV-MANO reference model.

Keywords: Network Function Virtualization, Computing In the Network, In-Network Computing, Network Virtualization

1 Introduction

NFV (Network Functions Virtualization) allows the replacement of traditional hardware-based middleboxes by Virtualized Network Functions (VNFs), which are executed on off-the-shelf hardware (Chiosi *et al.*, 2012). The advantages of NFV in comparison with the traditional alternative include greater flexibility and ease of management, at a lower cost and with less space and energy requirements (Han *et al.*, 2015; Mijumbi *et al.*, 2016). VNFs can be instantiated on demand, then used, and finally finished when they are no longer needed. Furthermore, VNFs can have their resources dynamically adjusted according to the demand, making efficient use of system resources. **Figure 1** illustrates a comparison between a traditional network and an NFV-based network. While hardware-based middleboxes make traditional networks more complex to manage, troubleshoot and deploy new services, NFV technology was developed as a way to address these issues.

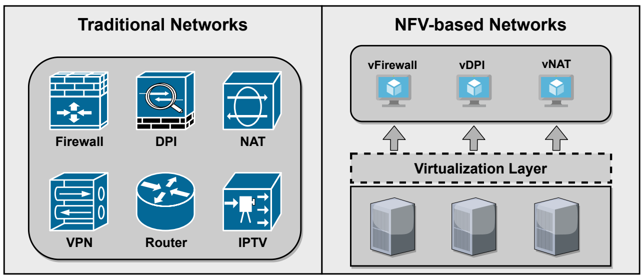

Figure 1. Comparison between a traditional network and an NFV-based network.

A single off-the-shelf server (hardware) is all that is needed to provide multiple services, instead of having to acquire/operate multiple servers (hardware), one per service. It is not necessary to get more hardware to provide new services. Although most VNFs implement classic middleboxes, such as firewalls or intrusion detectors, NFV technology can be used to provide novel, innovative services offered by the network itself. In this way, services that are normally executed on end-user hosts can be entirely executed within the network, in the paradigm that has been called Computing In the Network (COIN) (Zeng *et al.*, 2021). COIN is seen in the context of an edge-cloud continuum of applications and services, that seamlessly integrates in-network computing and network processing in a single framework. The very architecture of 5G and future 6G networks should benefit from COIN as an enabler of novel applications such as the Internet of Things, self-driving vehicles and distributed virtual reality.

It is not hard to anticipate the synergy of the union of NFV and COIN technologies. In this work besides discussing an NFV-COIN architecture, several case studies are presented. As early as 2015 (Turchetti and Duarte, 2015), an NFV-COIN service was implemented based on the classic failure detectors of distributed systems; that work was later expanded (Turchetti and Duarte Jr, 2017). Consensus was another distributed service implemented with NFV-COIN (Venâncio *et al.*, 2021), used to maintain the consistency of a group of SDN (Software-Defined Networks) controllers. In yet another effort, the network was augmented with the ability to natively offer reliable and ordered broadcast services (Venâncio *et al.*, 2019).

An alternative way to implement services in the network is to use programmable hardware (Tokusashi *et al.*, 2019). In 2016, Soulé et. al. (Dang *et al.*, 2016, 2020) implemented the Paxos consensus algorithm on a switch using P4. Other so called INC (In-Network Computing) services implemented with programmable hardware include data storage systems (Bressana *et al.*, 2020); an INC-based caching system called IncBricks (Liu *et al.*, 2017); and a distributed data partition/aggregation application (Sapio *et al.*, 2017).

Table 1. A summary of INC and NFV-COIN services.

Service	Reference/Year	NFV-COIN	Hardware INC	Performance
Failure Detector	(Turchetti and Duarte, 2015)	✓		Reduced CPU and memory usage
Failure Detector	(Turchetti and Duarte Jr, 2017)	✓		Reduced CPU and memory usage
Consensus	(Dang *et al.*, 2016)		✓	Increased throughput; Reduced latency
Consensus	(Dang *et al.*, 2020)		✓	Increased throughput
Consensus	(Venâncio *et al.*, 2021)	✓		Increased throughput; Reduced response time/CPU usage
Network Cache	(Liu *et al.*, 2017)		✓	Reduced request latency; Increased throughput
Aggregation	(Sapio *et al.*, 2017)		✓	Reduced bandwidth usage; Reduced computation time
Aggregation	(Lao *et al.*, 2021)		✓	Improved throughput; Efficient switch resource usage
Reliable Broadcast	(Venâncio *et al.*, 2019)	✓		Fair tradeoff between throughput and latency
Data Storage	(Bressana *et al.*, 2020)		✓	Increased throughput; Low latency overhead
Transaction Triaging	(Jepsen *et al.*, 2021)		✓	Increased throughput; Reduced computation time
Telemetry	(Misa *et al.*, 2021)		✓	Improved scalability with respect to traffic and query processing

Those services were implemented using technologies such as ASICs (Application Specific Integrated Circuits) and FPGA (Field Programmable Gate Array). In (Misa *et al.*, 2021) network telemetry operations are scheduled on programmable switches to improve the scalability of network-wide telemetry with respect to dynamic traffic and query loads. Telemetry traditionally relies in part on in-network band and computation (Marques *et al.*, 2019).

Compared to NFV-COIN, services implemented in hardware can present higher performance. On the other hand, NFV-COIN is certainly a better choice to deploy larger software systems that usually are executed on end-hosts. NFV-COIN inherits all the advantages of NFV over the hardware-based alternative: the major of which is certainly its flexibility to implement and manage new services. There is a reduction of both CAPital and OPerational EXpenditures (CAPEX & OPEX). It is also much easier to make new network services available: they can be simply published/downloaded from an Internet marketplace. To stop offering a service that is not needed anymore: just kill the corresponding processes. NFV-COIN can save energy, space, and computing resources. Nevertheless, we envision applications for both approaches.

The main difference between traditional SFCs and NFV-COIN services is that traditional SFCs are typical middlebox network services that have always been executed in the network. On the other hand NFV-COIN services are traditionally executed by end-users on their facilities, either as middleware or applications. Thus, from the point of view of end-users, the main advantage is the ease to simply use a service that is offered by the network itself, instead of having to implement or obtain and maintain/execute the service themselves. Some services can even be redesigned to use network information to provide new functionalities, an example is the NFV-FD failure detector mentioned above, which uses information exchanged by lower network layer protocols to detect link failures. **Table 1** presents a summary of in-network services developed either with programmable hardware or NFV.

The NFV-MANO reference architecture (NFV MANagement and Orchestration) (Quittek *et al.*, 2014) has been widely adopted as the standard to deploy and integrate virtual network services from different vendors and developers. NFV-COIN should be not only fully compliant with NFV-MANO, but also provide an interface to allow sophisticated types of interactions with end-user applications. It is necessary to precisely define how NFV-COIN services are offered, configured, maintained, and accessed. Thus an NFV-COIN architecture must include *(i)* an application interface that allows the end-user to invoke NFV-COIN services; on the other side, i.e. the network, it is necessary to have *(ii)* the system interface that allows access to NFV-COIN services; and, *(iii)* a system module to support the creation, operation and provision of NFV-COIN services, in addition to promoting secure access to those services.

The remainder of this article is organized as follows. Section 2 gives an overview of the NFV-MANO reference architecture. Section 3 introduces NFV-COIN and describes three case studies. Section 4 addresses research challenges and requirements for a generic NFV-COIN architecture. The conclusions follow in Section 5.

2 NFV-MANO: An Overview

In order to standardize the deployment, execution, and management of NFV-based services, as well as to allow the interoperability of a broad range of VNFs from different developers, the ETSI has coordinated efforts that led to the NFV-MANO (NFV Management and Orchestration) reference architecture (Quittek *et al.*, 2014). The MANO architecture is primarily responsible for the lifecycle, orchestration, and management of virtualized services. In addition, NFV-MANO provides standardized communication interfaces and abstracts the computational resources needed to execute VNFs, providing the support for running VNFs from different vendors (ETSI, 2015). **Figure 2** shows the main blocks of the reference architecture, which are described next: the NFV Infrastructure (NFVI), the VNFs, and the NFV-MANO block itself.

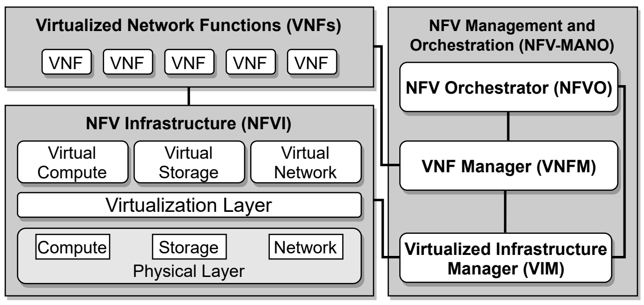

Figure 2. NFV-MANO reference architecture.

The NFVI block represents the virtualized infrastructure on which VNFs are instantiated, executed, and managed. The NFVI consists of a Physical Layer and a Virtualization Layer. The Physical Layer consists of computational resources, such as Compute, Storage, and Network. Physical resources are further abstracted as virtual elements through the Virtualization Layer, which is composed of a hypervisor that creates virtualized devices as Virtual Machines (VMs) or as containers. This allows each VNF to run independently and enforces isolation. The VNFs in **Figure 2** represent the virtual network function instances that are executing on the NFVI.

The VNFs block represents the instances of virtualized elements that support the execution of the network functions that are running on the NFVI. VNFs have well-defined communication and management interfaces. Each VNF has an associated VNF Descriptor (VNFD). A VNFD is a template responsible for specifying a VNF in terms of operational and deployment requirements (Quittek *et al.*, 2014). In addition to the VNFD, each VNF has a corresponding Element Management System (EMS). An EMS performs the typical FCAPS (Fault, Configuration, Accounting, Performance, Security) Management for one or several VNFs. This block also includes the SFCs – network services that are compositions of multiple VNFs.

The NFV-MANO block consists of three modules: NFVO, VNFM, and VIM, described next. The first module is the NFV Orchestrator (NFVO), which manages and orchestrates the resources allocated to the virtualized infrastructure. The NFVO also manages the lifecycle of network services, thus allowing the composition of VNFs on SFCs (Service Function Chains). In addition, the NFVO performs other tasks, such as increasing or decreasing computational resources according to demand, managing policies, validating and authorizing NFVI requests, among others.

The second module is the VNF Manager (VNFM), responsible for supervising the lifecycle of VNFs, including VNF instantiation, deletion, configuration, and scheduling. To perform its functionalities, the VNFM makes use of the VNFD. The VNFM is also responsible for the monitoring of VNF resources.

Finally, the Virtualized Infrastructure Manager (VIM) controls and manages the virtualized infrastructure resources (*i.e.* compute, storage, and network). The VIM, which typically is based on a cloud platform (*e.g.,* OpenStack (OpenStack, 2022)), is primarily responsible for providing and managing virtual resources. Some of the tasks performed by the VIM include: creation, deletion, and reconfiguration of virtual devices.

3 Computing In the Network with NFV

NFV technology has been used to support the creation and operation of services that run in the network and provide an interface for end-users. **Figure 3** represents a generic architecture of a virtualized network. The figure shows at the top a set of physical servers located in the network. These servers are endowed with a virtualization layer, which provides devices such as virtual machines and containers that abstract physical resources (*e.g.,* Compute, Storage, Network). Arbitrary network services can be built with VNFs running on this environment. The figure shows both examples of new NFV-COIN services and traditional network functions that have been usually deployed as VNFs, including firewalls, proxies, packet inspectors, among others. The NFV-COIN services are accessed by end-users from their networks shown at the bottom of the figure.

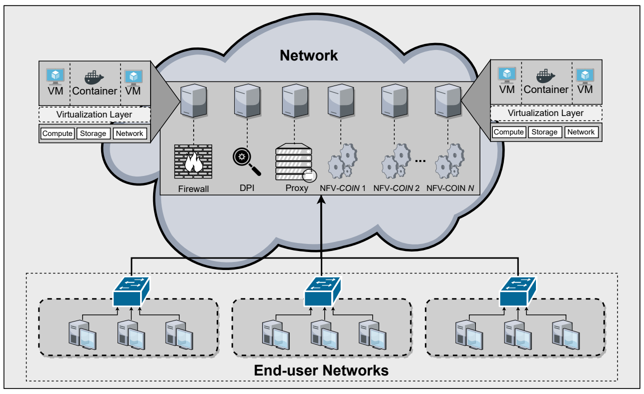

Figure 3. A generic virtualized network.

Next, we illustrate the potential of using NFV technology to build COIN services through three case studies. The first NFV-COIN service implements a failure detector. The second service implements consensus, applied to maintain the consistency of a distributed SDN control plane. Finally, the third case study presents a service that allows the network to offer reliable and ordered broadcast as native services.

3.1 Failure Detection as an NFV-COIN Service: NFV-FD

Failure detectors were originally proposed to help solve consensus in asynchronous distributed systems in which processes can fail by crashing (Chandra and Toueg, 1996). Asynchronous systems do not have timing guarantees, i.e. the time it takes for a message to reach the destination or to execute a task is not known in advance. Consensus is a fundamental problem of distributed systems that allows processes to agree on a single value, given a set of multiple initial values. Consensus must ensure agreement even if some processes fail. However, Lynch Fischer *et al.* (1985) and others proved the impossibility of consensus in asynchronous distributed systems with crash failures as early as 1985. In order to get around this impossibility, unreliable failure detectors were proposed as an abstraction to provide process state information (correct/suspect). Chandra and Toueg proved that, depending on the properties of the failure detector, consensus is possible in an asynchronous system with crash faults. Failure detectors are implemented by monitoring the processes. Usually, processes send *heartbeat* messages periodically. If the failure detector does not receive one (or more) of those messages within a suitable time interval, the process is suspected of having failed.

An NFV-based failure detector has been proposed (Turchetti and Duarte Jr, 2017) which is deployed as an NFV-COIN service. The solution, called NFV-FD (NFV Failure Detector) shown in **Figure 4**, uses information obtained from an SDN controller – in this case an OpenFlow controller (McKeown *et al.*, 2008) – to monitor processes and determine their status (*i.e.,* correct or suspected of having crashed). One of the main components of NFV-FD is a packet filter called FDMod, which is applied to the packets received by the controller, and selects packets that have information about the monitored processes. FDMod inspects packets by examining header fields and sends selected packets to NFV-FD that can update the status of the monitored processes.

NFV-FD also detects failures of communication links. For that, the OpenFlow topology discovery mechanism (Link Layer Discovery Protocol – LLDP) is used. The switches themselves send LLDP-type packets to inform any changes of the network topology. All LLDP packets are captured, filtered and processed by FDMod, which then sends the relevant information to the NFV-FD. For example, message *port s2-eth2 changed: DOWN* indicates that the *eth2* port of switch *s2* is down. NFV-FD in turn checks if there are processes connected through this port and marks those processes as unreachable.

Experimental results comparing NFV-FD with an alternative in which the failure detector is implemented in the controller itself (not as a standalone VNF) show that although the detection time of NFV-FD is slightly longer (about 120ms), there is a lower impact on the performance of the controller. When NFV-FD is executed within the controller, the CPU usage increases from 2.67% up to 4.49%. Therefore, placing the network function within the controller adds the corresponding resource requirements to the controller. Thus, the NFV approach allows the controllers to run only their native tasks, allowing a higher rate of flow requests.

3.2 Consensus as an NFV-COIN Service: VNF-Consensus

By default, the SDN control plane is centralized, which raises issues in terms of availability, scalability and performance (Canini *et al.*, 2015). Although a distributed control plane can

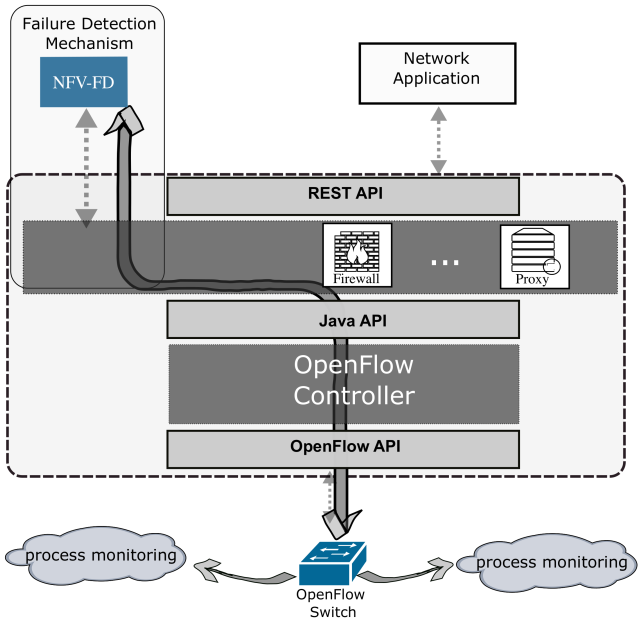

Figure 4. NFV-FD architecture.

solve the problem, achieving consistent synchronization between multiple SDN controllers is not a trivial task. Existing solutions employ the controllers themselves to perform the synchronization tasks, or alternatively synchronize the data plane. In (Venâncio *et al.*, 2021) the VNF-Consensus service is proposed to implement the Paxos consensus algorithm and to maintain the consistency of a distributed control plane that consists of multiple SDN controllers. VNF-Consensus does not require any modification of the data plane or the SDN protocol (in this case, again OpenFlow).

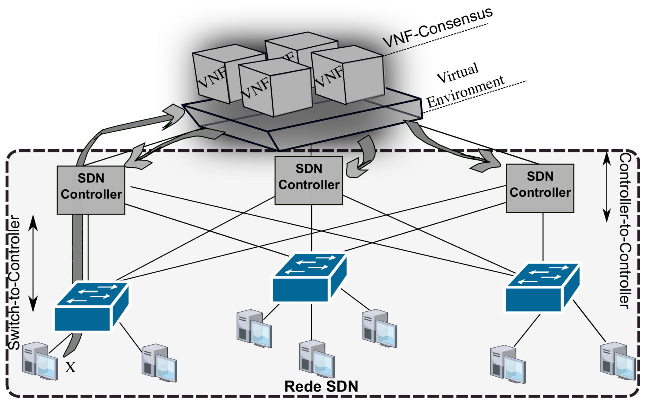

Figure 5. VNF-Consensus architecture.

Figure 5 shows how VNF-Consensus works. To ensure the consistency of the distributed control plane, before a new OpenFlow rule is installed on some switch of the data plane, it is replicated to all SDN controllers. The controller that first receives the new rule starts VNF-Consensus which executes the Paxos algorithm to guarantee that all controllers get the rule. The data plane can then be consistently updated with that rule. In this way, if any controller fails, any other controller can take over, since they have exactly the same view of the network.

Experimental results show that using VNF-Consensus is more efficient than having the controllers themselves run consensus to guarantee the consistency of the distributed control plane. When using VNF-Consensus, the controller CPU usage drops from 62.1% to around 34.4%. Hence, the controller is able to handle a larger number of requests: around 53% more. At the same time, the latency to install a consistent rule over multiple controllers decreases significantly: up to 18.5%. Finally, when using the VNF approach, the throughput is about 3.6 times higher than the corresponding classic strategy when comparing with scenarios under heavier loads.

3.3 Reliable and Ordered Broadcast as an NFV-COIN Service: NFV-RBCast

Reliable broadcast is one of the most important abstractions for the development of distributed systems Hadzilacos and Toueg (1993). A source process can use reliable broadcast to send a message that is guaranteed to be correctly delivered by all correct processes, even if the source fails. Other broadcast abstractions also guarantee the *order* in which messages are delivered. The most common ordered broadcast abstractions are FIFO (First-In First-Out), causal and atomic broadcast. FIFO broadcast guarantees that messages are delivered according to the order in which they were broadcast by the source process. Causal broadcast guarantees the causality of message delivery. Finally, atomic broadcast guarantees that all messages are delivered by all correct processes in the same total order. The orders can be combined, so it is possible for example to have atomic FIFO broadcast or causal atomic broadcast.

Traditionally, end-users either implement broadcast within the distributed applications that need this type of service or, alternatively install/maintain middleware with this functionality. The NFV-RBCast (Venâncio *et al*., 2019) solution uses NFV-COIN technology to allow the network itself to provide reliable and ordered broadcast as native services. NFV-RBCast currently features reliable broadcast, atomic broadcast, FIFO atomic broadcast and causal atomic broadcast. Applications access the broadcast primitives through an API called *RBCast*.

NFV-RBCast guarantees the order in which messages are delivered by using a sequencer which is implemented as a VNF, called VNF-Sequencer, shown in **Figure 6**. From the standard operation of the OpenFlow protocol, whenever a switch receives a packet that does not have a matching entry in the flow table, the packet is forwarded to the controller using a *packet_in* message. If the *packet_in* message indicates that the flow is of the NFV-RBCast service, the controller installs a rule in the switch so that all packets of this flow are forwarded to the VNF-Sequencer. Upon receiving those packets, the VNF-Sequencer executes the sorting algorithm specified. VNF-Sequencer inserts a sequence number into each packet, which is next forwarded to the destination processes. Upon receiving a broadcast packet, the destination process delivers the message to the application taking into account the sequence number inserted by VNF-Sequencer.

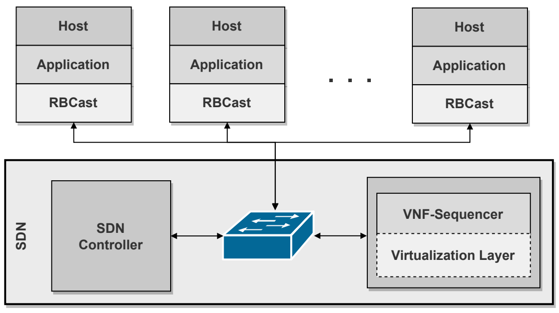

Figure 6. NFV-RBCast architecture.

Experimental results show that VNF-Consensus achieves a throughput of up to 15.000 messages/second for atomic broadcast executed by 50 processes. The delivery latency of atomic broadcast was up to 32% higher than that of reliable broadcast, which is expected since atomic broadcast guarantees the total order of messages.

4 An NFV-COIN Architecture

In this section we discuss the requirements of an architecture to support NFV-COIN services. NFV-COIN leverages NFV technology to allow the deployment of COIN services. It is important to keep in mind that these are network native services they are meant to be accessed by end-users. As shown in **Figure 7**, the architecture must include a User Module, which runs on the end-user host. This module must provide an API for end-users to access the NFV-COIN services, called user NFV-COIN API. This API can be implemented as a library with primitives to allow access and management of the NFV-COIN services, enabling their creation, configuration, usage, monitoring and deletion. The NFV-COIN API must also allow users to integrate multiple services to their local applications in a transparent and unified way.

Figure 7 also shows the NFV Module which runs in the network and consists of a traditional NFV system, plus an NFV-COIN Supervisor (NFVS). The NFVS can be seen as a gateway to the NFV-COIN services, and consists of a Broker, a Configuration Agent and Monitoring Module, described below. Furthermore, the NFV-COIN Services API provides the interface that allows access to the NFVS and NFV-COIN services within the network. The User Module communicates with the NFV Module through the interfaces provided by the NFV-COIN API that runs on an end-user host and NFV-COIN Services API in the network.

As an example of these APIs, the NFV-FD failure detector must include in the NFV-COIN service API a primitive to output the list of processes suspected of having crashed, while the NFV-COIN user API must have a primitive to invoke that function. Other NFV-FD API primitives include those for adding and removing processes of the list of processes to be monitored. The VNF-Consensus user API must include primitives to propose and decide values. The NFV-RBCast user API must have primitives for the different types of broadcasts supported, such as for atomic broadcast or for reliable broadcast, among others. There is also a primitive to deliver a message to the end-user application.

The NFV-COIN Services API thus allows end-user applications to access the NFV-COIN services through the User API. Thus both APIs could be implemented for instance as REST (REpresentational State Transfer) interfaces, that make it easy to integrate heterogeneous applications and systems from different vendors. As shown in the examples above, the NFV-COIN Services API consists of sets of primitives which are defined case-by-case for each NFV-COIN service, there is no general set of primitives that can be adopted for all services. Thus the NFV-COIN Services API should also provide a discovery service, returning to the end-user a catalog with the NFV-COIN services available. The Services API must be generic enough in the sense that: *(i)* it allows end-user applications to access the NFV-COIN services available on the network through standard calls; *(ii)* the API should support different types of virtualization technologies, such as NFV, SDN, and also cloud computing; *(iii)* the API must be compatible with most programming languages, allowing interoperability between different developers.

The *NFV Module* in turn consists of the original components of the NFV-MANO architecture (*i.e.,* NFVO, VNFM, VIM), and also includes the physical infrastructure on which services are virtualized (*i.e.,* the NFVI), the virtualized services themselves and also the NFVS (NFV-COIN Supervisor) mentioned above. The NFVS is the main component of the NFV-COIN architecture, and its functionalities can be organized in three different modules: the Broker, Configuration Agent and Monitoring Module, in addition to the afore-

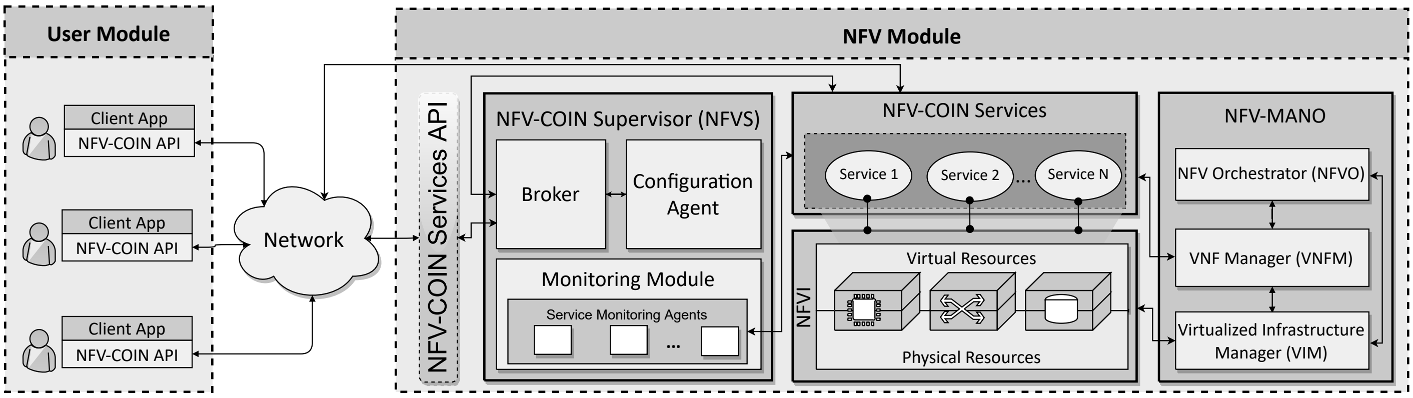

Figure 7. An NFV-COIN architecture.

mentioned NFV-COIN Services API.

The NFVS acts as a gateway between the end-user (the application, actually), the virtualized services and the virtualization layer. All requests made by the application through the user NFV-COIN API are received by the NFV-COIN Services API. The NFV-COIN Services API forwards requests to the Broker, which is responsible for parsing requests and establishing the traffic between the application and the corresponding NFV-COIN services.

The Broker is responsible for receiving and processing requests sent by the NFV-COIN Services API. Requests can be then forwarded to the NFV-MANO system for accomplishing tasks such as instantiating a new NFV-COIN service, or monitoring, deleting or making some reconfiguration of an existing service. Requests can also be scheduled for future execution when they cannot/should not be executed instantly.

The NFVS must also have a Configuration Agent (CA), responsible for both the general configuration of the NFV-COIN services and the configuration of each user session, including guaranteeing that the communication between each user and service is secure. Users must be authorized and all communication with NFV-COIN services must be authenticated and confidentiality must be guaranteed as well. In addition, the CA creates a dedicated virtual network for each user, allowing individual instances of the virtualized services to be created. This approach ensures isolation, thus privacy and security; in addition, as there is no competition for resources, it prevents bottlenecks from arising even when the number of users increases. The CA also maintains information about users and their respective connections on a local database, including *e.g.,* the services each user is connected to, which virtual network is associated with which user and the respective active connections.

Finally, the NFVS also features a Monitoring module, to monitor the state of instantiated services, also providing information about the system itself, including the NFVI and the communication channels. The Monitoring module must support arbitrary metrics such as CPU utilization, bandwidth (in/out) and memory utilization. Note that this module must be able to monitor NFV-COIN services that consist of multiple geographically separated functions, as well as the underlying network. Thus *e.g.* it should be able to detect network faults that can lead to NFV-COIN service failures.

4.1 NFV-COIN Architecture: Operations

Next we describe operations of the NFV-COIN architecture to allow NFV-COIN services to be provided, managed and accessed: *(i)* operations for the creation of secure, authenticated connections between end-users and NFV-COIN services; *(ii)* operations related to the communication of end-users and NFV-COIN services; *(iii)* operations related to the management of NFV-COIN services. These different types of operations are described next.

Connecting with NFV-COIN Services

After an application invokes an NFV-COIN service, a session is created between user and service. Before that, the user may have called the *discovery* function of NFV-COIN Services API, in order to obtain a list of the services available on the network. A connection request is forwarded to the Broker, which sends a message to the CA module to establish a secure connection between user and service.

In the first step in which a new session is created, the CA makes a request to the NFV-MANO system to set up a new virtual network for the user. This ensures isolation and security. The second step consists of the authentication of both parts. Finally, in the third step, the Broker registers the session on the CA local database. After the session is created, the Broker makes a request to instantiate the invoked NFV-COIN services. To do so, NFV-MANO creates a new instance of the required service on the user's virtual network. The Broker waits until the instantiation has completed, then returns a message to the application informing that the service is available.

Communicating with NFV-COIN Services

After the connection is established, the end-user application can communicate with the NFV-COIN service either directly or through the NFV-COIN Services API, a choice that depends mainly on performance issues. Although the direct communication is more efficient, the NFV-COIN Services API can be more convenient as it provides a REST-like interface that provides a set of primitives that can be obtained as a result of the discovery call. On the other hand, the direct communication can be the best choice for instance when high volumes of data must be transferred. The direct communication is established by the Broker. The application communicates using the proper NFV-COIN user API primitives. The Broker also needs to create the routes across the end-user virtual network so that the application can reach the NFV-COIN service.

Managing the NFV-COIN Services

The NFV-COIN Services API provides a number of operations for service management. Examples include operations for reconfiguration, such as auto-scaling that allows a service to have its assigned resources or even the number of running instances increased or reduced depending on the load. Some operations involve other modules of the architecture, others are executed directly on the virtualized service.

Note that the Broker is concerned with the services – not the VNFs, which are a responsibility of NFV-MANO itself. Thus the Broker is not responsible for the lifecycle of the corresponding VNFs/SFCs, but for the management of the NFV-COIN services themselves. For instance, consider the failure detector service (NFV-FD) described above. The Broker is responsible for performance monitoring and, depending on the number of processes being monitored by NFV-FD its assigned resources can be increased and/or decreased.

5 Conclusion

This article explores a novel alternative for the provision of arbitrary services within/by the network (*i.e.,* COIN: Computing In the Network) based on NFV technology. The main difference of NFV-COIN services from traditional virtualized network services is that they are invoked by end-user applications through primitives provided by an API. In the article, before discussing an NFV-COIN architecture, three NFV-COIN services were presented as case studies: NFV-FD, VNF-Consensus and NFV-RBCast. These services have already been implemented and clearly demonstrate the feasibility of deploying COIN services with NFV technology.

An NFV-COIN architecture is described as an extension to the NFV-MANO reference model that allows services to be easily invoked/integrated by end-user applications. The architecture consists four main components: *(i)* an application API, with primitives to allow the end-user to invoke an NFV-COIN service; *(ii)* the NFV-COIN API, which runs in the network and provides the interface from requests to services; *(iii)* the NFVS, the NFV-COIN Supervisor, a system module to support the creation, operation and provision of NFV-COIN services, in addition to promoting secure access to those services; *(iv)* the NFV-COIN services themselves.

There are multiple challenges that must be tackled until NFV-COIN can become a reality in production networks. Although we discussed an architecture in the paper, much work has to be done until a complete, unifying architecture that supports highly different NFV-COIN services is available. Performance is always an issued for systems based on multiple software layers. Although there are alternatives for example in hardware, pursuing the implementation of performant NFV-COIN services is a formidable challenge that can bring significant advantages. Last but not least, dependability issues must also be solved. NFV-COIN services will only be widely adopted on production networks if they can provide the availability levels required. However, there is no doubt that leveraging networks to offer arbitrary services on demand which are built with virtualization technologies is worth investigating.

Acknowledgements

This work was partially supported by the Brazilian Research Council (CNPq) grant 308959/2020-5, FAPESP/MCTIC/CGI 2021/06923-0 and CAPES ”Código de Financiamento” 001

References

Bressana, P., Zilberman, N., Vucinic, D., and Soulé, R. (2020). Trading latency for compute in the network. In *Proceedings of the 2020 Workshop on Network Application Integration/CoDesign, NAI@SIGCOMM 2020, Virtual Event, USA, August 14, 2020*, pages 35–40. ACM. DOI: 10.1145/3405672.3405807.

Canini, M., Kuznetsov, P., Levin, D., and Schmid, S. (2015). A distributed and robust SDN control plane for transactional network updates. In *IEEE Conference on Computer Communications (INFOCOM)*. DOI: 10.1109/INFOCOM.2015.7218382.

Chandra, T. D. and Toueg, S. (1996). Unreliable failure detectors for reliable distributed systems. *Journal of ACM*, 43(2). DOI: 10.1145/226643.226647.

Chiosi, M., Clarke, D., Willis, P., Reid, A., Feger, J., Bugenhagen, M., Khan, W., Fargano, M., Cui, C., Deng, H., *et al.* (2012). Network functions virtualisation: An introduction, benefits, enablers, challenges and call for action. Available at: `http://course.ipv6.club.tw/SDN/nfv_white_paper.pdf`.

Dang, H. T., Bressana, P., Wang, H., Lee, K. S., Zilberman, N., Weatherspoon, H., Canini, M., Pedone, F., and Soulé, R. (2020). P4xos: Consensus as a network service. *IEEE/ACM Transactions on Networking*. DOI: 10.1109/TNET.2020.2992106.

Dang, H. T., Canini, M., Pedone, F., and Soulé, R. (2016). Paxos made switch-y. *ACM SIGCOMM Computer Communication Review*, 46(2):18–24. DOI: 10.1145/2935634.2935638.

ETSI, N. (2015). Network functions virtualization (nfv) infrastructure overview. Avilable at: `https://www.etsi.org/deliver/etsi_gs/nfv-inf/001_099/001/01.01.01_60/gs_nfv-inf001v010101p.pdf`.

Fischer, M. J., Lynch, N. A., and Paterson, M. S. (1985). Impossibility of distributed consensus with one faulty process. *Journal of the ACM (JACM)*, 32(2):374–382. DOI: 10.1145/3149.214121.

Hadzilacos, V. and Toueg, S. (1993). Fault-tolerant broadcasts and related problems. In *Distributed systems (2nd Ed.)*, pages 97–145. Addison-Wesley. Available at: `https://dl.acm.org/doi/abs/10.5555/302430.302435`.

Han, B., Gopalakrishnan, V., Ji, L., and Lee, S. (2015). Network function virtualization: Challenges and opportunities for innovations. *IEEE Communications Magazine*, 53(2):90–97. DOI: 10.1109/MCOM.2015.7045396.

Jepsen, T., Lerner, A., Pedone, F., Soulé, R., and Cudré-Mauroux, P. (2021). In-network support for transaction triaging. *Proceedings of the VLDB Endowment*, 14(9):1626–1639. DOI: 10.14778/3461535.3461551.

Lao, C., Le, Y., Mahajan, K., Chen, Y., Wu, W., Akella, A., and Swift, M. (2021). ATP: In-network aggregation for multi-tenant learning. In *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*, pages 741–761. Available at: `https://www.usenix.org/system/files/nsdi21-lao.pdf`.

Liu, M., Luo, L., Nelson, J., Ceze, L., Krishnamurthy, A., and Atreya, K. (2017). Incbricks: Toward in-network computation with an in-network cache. In *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 795–809. DOI: 10.1145/3037697.3037731.

Marques, J. A., Luizelli, M. C., Tavares da Costa Filho, R. I., and Gaspary, L. P. (2019). An optimization-based approach for efficient network monitoring using in-band network telemetry. *Journal of Internet Services and Applications*, 10(1):1–20. DOI: 10.1186/s13174-019-0112-0.

McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., Shenker, S., and Turner, J. (2008). Openflow: enabling innovation in campus networks. *ACM SIGCOMM computer communication review*, 38(2):69–74. DOI: 10.1145/1355734.1355746.

Mijumbi, R., Serrat, J., Gorricho, J. L., Bouten, N., Turck, F. D., and Boutaba, R. (2016). Network Function Virtualization: State-of-the-Art and Research Challenges. *IEEE Communications Surveys Tutorials*, 18(1). DOI: 10.1109/COMST.2015.2477041.

Misa, C., Durairajan, R., Rejaie, R., and Willinger, W. (2021). Revisiting network telemetry in coin: A case for runtime programmability. *IEEE Network*, 35(5):14–20. DOI: 10.1109/MNET.201.2100064.

OpenStack (2022). Openstack. `https://www.openstack.org/`. Accessed: 2022.

Quittek, J., Bauskar, P., BenMeriem, T., Bennett, A., Besson, M., and et al (2014). Network Functions Virtualisation (NFV); Management and Orchestration. GS NFV-MAN 001 V1.1.1. Available at: `http://www.etsi.org/deliver/etsi{_}gs/NFV-MAN/001{_}099/001/01.01.01{_}60/gs{_}nfv-man001v010101p.pdf`.

Sapio, A., Abdelaziz, I., Aldilaijan, A., Canini, M., and Kalnis, P. (2017). In-network computation is a dumb idea whose time has come. In *Proceedings of the 16th ACM Workshop on Hot Topics in Networks*, pages 150–156. DOI: 10.1145/3152434.3152461.

Tokusashi, Y., Dang, H. T., Pedone, F., Soulé, R., and Zilberman, N. (2019). The case for in-network computing on demand. In *Proceedings of the Fourteenth EuroSys Conference 2019*, pages 1–16. DOI: 10.1145/3302424.3303979.

Turchetti, R. C. and Duarte, E. P. (2015). Implementation of failure detector based on network function virtualization. In *2015 IEEE International Conference on Dependable Systems and Networks Workshops*, pages 19–25. IEEE. DOI: 10.1109/DSN-W.2015.30.

Turchetti, R. C. and Duarte Jr, E. P. (2017). Nfv-fd: Implementation of a failure detector using network virtualization technology. *International Journal of Network Management*, 27(6):e1988. DOI: 10.1002/nem.1988.

Venâncio, G., Turchetti, R. C., Camargo, E. T., and Duarte Jr, E. P. (2021). Vnf-consensus: A virtual network function for maintaining a consistent distributed software-defined network control plane. *International Journal of Network Management*, 31(3):e2124. DOI: 10.1002/nem.2124.

Venâncio, G., Turchetti, R. C., and Duarte, E. P. (2019). Nfv-rbcast: Enabling the network to offer reliable and ordered broadcast services. In *2019 9th Latin-American Symposium on Dependable Computing (LADC)*, pages 1–10. IEEE. DOI: 10.1109/LADC48089.2019.8995681.

Zeng, D., Ansari, N., Montpetit, M.-J., Schooler, E. M., and Tarchi, D. (2021). Guest editorial: In-network computing: Emerging trends for the edge-cloud continuum. *IEEE Network*, 35(5):12–13. DOI: 10.1109/MNET.2021.9606835.