Abstract:

Network operators are often very cautious before deploying any novel networking service. This is done only if the new networking solution is fully monitored, secured and can provide rapid return on investments. By adopting the emerging Network Functions Virtualization (NFV) concept, network operators will be able to overcome this constraint by allowing them to deploy solutions at lower costs and risks. Indeed, NFV involves implementing network functions in software that can rely on virtualization techniques to run on standard server hardware, and that can then be deployed in, or moved to, various network location as required. This document analyzes and assesses how to leverage IT virtualization and determine which solutions are the most appropriate in the DOCTOR project to design a flexible NFV-based architecture that can host new networking services, such as the NDN content delivery service, in virtualized environments. We also present the different requirements and challenges for the monitoring and security issues, making it possible to efficiently secure the overall virtualized architecture.
TABLE OF CONTENTS

1 Introduction ........................................................................................................................................4
2 Named Data Networking ...........................................................................................................................6
  2.1 Naming Scheme ...................................................................................................................................6
  2.2 Packet Structure ...................................................................................................................................6
  2.3 Content Routers ..................................................................................................................................7
  2.4 The Matching Rule in NDN ...................................................................................................................7
    2.4.1 Forwarding Strategy Module ...........................................................................................................9
  2.5 NDN Transport ......................................................................................................................................10
  2.6 Basic Security Mechanisms ..................................................................................................................10
3 Approaches of Virtualization ..................................................................................................................11
  3.1 Definitions ..........................................................................................................................................11
  3.2 Types of Virtualization Technology ......................................................................................................12
    3.2.1 User-Mode Hosted Virtualization (Type-2 Virtualization) .................................................................14
    3.2.2 Native or Bare-Metal Virtualization (Type-1 Virtualization) ............................................................15
    3.2.3 OS-Level Virtualization (Type-0 Virtualization) ...............................................................................19
  3.3 Vertical and Horizontal Virtualization .................................................................................................21
  3.4 Virtualization in Data Centers .............................................................................................................22
    3.4.1 Network Interface Virtualization .....................................................................................................22
    3.4.2 Virtualized Components ..................................................................................................................24
  3.5 OpenStack Architecture .......................................................................................................................25
    3.5.1 OpenStack Networking ...................................................................................................................26
    3.5.2 Virtual Networking Components and Namespaces .........................................................................27
4 Towards Network Function Virtualization ...............................................................................................28
  4.1 The NFV Architecture and Technical Foundations ..............................................................................28
    4.1.1 The NFV Paradigm ..........................................................................................................................28
    4.1.2 The NFV Architectural Framework .................................................................................................29
    4.1.3 NFV POCs ........................................................................................................................................30
    4.1.4 OPNFV ............................................................................................................................................31
    4.1.5 The OpenStack Effort for NFV ......................................................................................................31
   4.1.6 Enabling NFV by Providing High Performance in Virtualized Networks .........................................31
  4.2 NFV Hand and Hand with SDN .............................................................................................................32
5 Requirements and Challenges for DOCTOR .........................................................................................36
  5.1 Requirements and Challenges for Monitoring Network Traffic in Virtualized Environments ............36
    5.1.1 Definition and Scope of Monitoring ...............................................................................................36
# Deliverable D1.1: Virtualization Techniques: Analysis and Selection

5.1.2 Monitoring in Virtualized Environments ..........................................................37
5.1.3 Where to Implement Monitoring Functions ......................................................39
5.1.4 SDN-based Monitoring Architecture ..............................................................40
5.1.5 Coordinating Security and Traffic Management .............................................41
5.1.6 Requirements and Features of the SDN-based Monitoring System ...............41

5.2 Requirements and Challenges for Risk Assessment in Virtualized Environments ....43
5.2.1 Risk Assessment ..............................................................................................43
5.2.2 Risk Assessment Models ................................................................................44
5.2.3 Risk Assessment Models Inputs ......................................................................45
5.2.4 Risk Assessment Architectures ........................................................................45
5.2.5 Challenges and Advantages of Risk Assessment in Virtualized Environments ...47

5.3 Requirements and Challenges of Remediations Deployment in Virtualized Environments 47
5.3.1 Remediations ....................................................................................................47
5.3.2 Types of Remediation ....................................................................................48
5.3.3 Simulation of the Network Topology ..............................................................48
5.3.4 Challenges and Advantages of the Deployment of Remediations in Virtualized Environments .................................................................48

6 Assessment of the Requirements and Possible Candidates ...................................50
6.1 Infrastructure Layer ............................................................................................51
6.2 Virtualization Layer ..........................................................................................51
6.3 Application Layer ..............................................................................................51
6.4 Control Layer .....................................................................................................52

7 Conclusion ............................................................................................................53

8 References ............................................................................................................54
1 Introduction

Network equipment is generally implemented as special-purpose appliances built over proprietary software platforms tied onto proprietary hardware. This approach involves operational challenges and high costs of managing issues such as scaling up networking resources to address increasing users’ demands and the equipment’s end-of-life. Moreover, it obstructs innovation and prevents new networking services from being deployed to support changing business requirements. For example, global network traffic has increased considerably over the past decade, and will continue to rapidly evolve over the next years, mainly due to new usage patterns on Internet, which are more and more related to content dissemination and consumption. In response to this trend, there is a significant consensus, in the research community, of the benefits of Named Data Networking (NDN) [1], a new routing paradigm that shifts the thin waist of the Internet from the today’s host-centric architecture to a content-centric model. Yet network operators are still reluctant to push such innovation into their production networks.

As server virtualization had gained popularity and maturity in IT data centers, the constraints of hardware-based appliances have led operators, in late 2012, to apply standard IT virtualization technologies to their networks, thus defining the basis foundation for Network Functions Virtualization (NFV) in an Industry Specification Group (ISG) within the European Telecommunications Standards Institute (ETSI). Since then, the NFV initiative has generated a great deal of interest and is steadily striving to design and implement network functions entirely in software running on industry-standard hardware. Examples of such network functions are network address translation, traffic analysis ( DPI), load balancers, caching, firewalling and intrusion detection, to name a few. The virtualization of network functions over shared pools of standardized commodity hardware resources will enable to achieve greater agility, flexibility and elasticity for network configuration and operation, resulting in reduced time to market for deploying new networking service while reducing both operational (OPEX) and equipment costs (CAPEX).

The DOCTOR project proposes to design a flexible virtualized and service-driven network architecture for easing the deployment of new networking services or protocols. While advocating the adoption of the NFV principles, the DOCTOR virtualized architecture abstracts the network layers from hardware (i.e. removing any dependence on proprietary hardware) and implements network functions and service applications as software modules that can be deployed on standard computing hardware. Leveraging the NFV allows thus for agile and elastic placement of networking services when and where they are needed. We also integrate in our proposed virtualized architecture a management layer that provides the ability to program the behaviour of the network, based on the Software-Defined Networking (SDN) concept to separate the data plane from the control plane. This management layer enables a centralized orchestration to dynamically configure routing data in the network and offers network monitoring functions for performance analysis and defense against security threats in virtualized environments. Indeed, monitoring and security are important requirements that network operators need to address before deploying any new service. To illustrate our solution, we consider the case of NDN as a new emerging network protocol.

We present in this document how and which virtualization techniques will offer the adequate capabilities, with respect to the NFV architectural guidelines, for designing the DOCTOR infrastructure to host new functions and services which can be secured and perform at high throughput, having in mind the use case of NDN deployment. The document is then organized as follows.

Section 2 of the document reminds the NDN principles. Section 3 proposes a description of existing server virtualization techniques and their application to IT data centers, using for example the emerging open-source OpenStack framework for horizontally scaling virtual resources over a cluster of hardware entities. In Section 4, we present the Network Function Virtualization (NFV) and the Software-Defined Networking (SDN) paradigms and introduce how both these technologies can collaborate. In Section 5, we analyze the requirements and challenges for monitoring network stacks deployed in a virtualized environment, regarding to the type information to monitor, the way to collect it and the way to analyze it, while capitalising on the full potential of the SDN concept. This monitored data is then useful for detecting and mitigating security threats related to both IP and
NDN that can be challenging in virtual networks. Leveraging a virtualized networking technology requires a full rethought of the way the security has to be designed, implemented and orchestrated. We also focus on risk assessment and remediations deployment for our use case NDN deployed in our virtualized architecture. Section 6 then analyzes and assesses those requirements for network monitoring and security functions deployment. Finally, Section 7 concludes the document and outlines our next steps in the project.
2 Named Data Networking

The Content Centric Networking (CCN) architecture from PARC is a pioneering fully-fledged Information-Centric Networking (ICN) architecture [2][3]. Its basic ideas were described in a Google tech talk, long before the first paper describing the CCN architecture was published [4]. The Named Data Networking (NDN) project [1], funded by the US Future Internet Architecture program, is further developing the CCN architecture. A crucial aspect of NDN is that names are hierarchical, thus allowing name resolution and data routing information to be aggregated across similar names, something considered to be critical for the scalability of the architecture.

2.1 Naming Scheme

NDN uses hierarchical, variable-length names. It is composed of a number of components. For an easy understanding, NDN hierarchical names are presented under the form of URIs with “/” characters separating components. Figure 1 is an example. In order to allow data providers and data consumers to operate over NDN content names, a name convention must be agreed by both of them. In addition, naming contents independently between providers cannot avoid creating duplicate name for different contents. Therefore, a naming system is required in order to define and allocate top-level names, ensuring the uniqueness of content names. Such system remains an open challenge and still under active research. However, not all NDN names need to be globally unique. In some local communications, NDN names can be based on local context.

![Figure 1: Name Structure in NDN](image)

A data file in NDN is equally divided into segments, each one also referred to as Named Data Object (NDO). Besides, a provider often wants to provide its up-to-date content. Hence, a NDN name also includes information about version and segment numbers. In the NDNx project, the version component is the next-to-last component and based on timestamp. A NDNx timestamp is expressed with 48 bits in binary encoding, in units of 2−12 seconds since the start of Unix time (i.e. Jan 1, 1970). The last name component stands for the segment number. The segment number is limited to 7 bytes. NDN’s name can support both consecutive and non-consecutive numbering of segment. For example, the name in Figure 1 indicates that the content is the 2000th segment of the file UTTtrailer.mpg which is updated in Feb 13, 2009 at 15:31:30 (1234567890 seconds since Jan 1, 1970).

2.2 Packet Structure

There are two NDN packet types: Interest and Data. A consumer asks for content by sending an Interest to the network through an appropriate interface. An Interest packet consists of these fields:

- **Content name**: the NDN name of the requested Named Data Object (NDO);
- **Selectors**: contains preferences for selection in case there are more than one matching content object found at a location. These preferences can be components that should not appear in the name of the returned object; whether an older version or a latest version is more preferred;
- **Nonce**: a random value that helps discard duplicate Interests. By comparing the nonce value of Interest packets that request for the same Content name, but come from different interfaces, a network element can eliminate duplicate Interest packets;
- **Guiders**: includes information about Scope and Interest lifetime. Scope limits the propagation range of Interest. Scope 0 limits the Interest propagation inside an application on a host. Scope 1 allows Interest to be exchanged between applications in the same host. Scope 2 re-
fers that propagation is no further than the next host (not the next router). Other values are not defined and will cause the Interest to be dropped. Interest Lifetime indicates the time remaining before the Interest times out. This field is set by applications.

Any node hearing the Interest and having the requested data can respond with a Data packet. A Data packet has 4 fields:

- **Content Name**: the NDN name of the requested NDO whose data is included in the Content field of the packet;
- **MetaInfo**: includes supporting information about the content, such as content type, freshness period;
- **Content**: contains a piece of the requested information (or the whole information, depending on the granularity);
- **Signature**: this field holds information to ensure the integrity and authenticity of the content field, e.g. signature type, key locator, signature bits. While signature type indicates the algorithm used to compute the signature bits, key locator will provide guides to retrieve the publisher's public key for the verification process. In order to yield the signature bits, first, the provider computes hash over the Content name, the Signed Info and Content fields. Then, the hash value is encrypted, using provider's private key. The result will be the digital signature bits in this field. In addition, this field also indicates the algorithms used to compute the digital signature bits.

### 2.3 Content Routers

Name resolution and data routing functionalities in NDN are integrated. The Content Routers (CRs) are network components that have responsibility for these two functionalities. CR has 3 main data structures:

- **Forwarding Information Base (FIB)**: similar to IP’s routing table. Each FIB entry consists of a NDN name prefix and a list of outgoing interfaces that will be used to forward Interest toward content providers. For now, there is no concrete routing protocol in NDN. For the initial launch of NDN, the authors of this architecture propose to extend the implementation of the OSPF (a.k.a OSPFN, for intra-domain) and BGP (for inter-domain) routing protocols to support name prefixes and multipath forwarding of Interests [1]. Recently, the Named-data Link State Routing (NLSR) protocol [5], built directly over the NDN stack, has been proposed for intra-domain, using Interest and Data messages to carry route announcements and a hop-by-hop synchronization protocol as an efficient replacement of the traditional network-wide flooding of OSPFN for keeping routers updated on the name-based network topology.
- **Pending Interest Table (PIT)**: used to forward Data packet back to requesters. Each PIT entry consists of a name prefix, a list that contains all incoming interfaces of unsatisfied Interests for that prefix and their nonce values.
- **Content Store (CS)**: a local cache in CR, provides on-path caching in NDN. Suggested caching policies for CS is Least Recently Used (LRU) and Least Frequently Used (LFU).

### 2.4 The Matching Rule in NDN

Match rule in NDN is divided in two cases. The first case is to match a content name to FIB/PIT entries. Name in a matching entry must be prefix of the considered name. For easy understanding, assume that an Interest requests for a content name /A/B/C.jpg and there are 2 entries for /A and /A/B in a FIB table. Both 2 entries are prefix matches to /A/B/C.jpg but the /A/B entry is the only longest-prefix matching entry and is chosen. However, none of these 2 entries is the exact-match entry. The only possible exact-match entry is /A/B/C.jpg. Length of the considered name is greater than or equal to length of the entry's name, since routing table usually keeps aggregated information in order to reduce its size.

The second case is to match a content name to content objects (including objects in CS or in provider). The content name must be prefix of matching objects. For easy understanding, assume that an Interest requests for a content name /A/B/C and there are 4 objects that need to be considered: /A/D.jpg/9999/15, /A/B/E.jpg/3691/11, /A/B/C/F.jpg/1234/0, /A/B/C/G/H.txt/3456/1. Two first
objects do not match for the requested name, since /A/B/C is not their prefixes. Both two last objects are matches for the requested name. Exact-match is also allowed in the second case. Length of the considered name is less than or equal to the length of matching objects' name, since in most case, the full name of Data is not known precisely. The consumer usually specifies it relative to something whose name is known.

Figure 2 is a block diagram of CR's operation when it receives an Interest and a Data and Figure 3 demonstrates the path of packets in NDN. The Interest mentioned in the example is sent by User 1 for a content name /doctor/index.htm and User 2 is assumed to send an Interest after User 1 received the Data packet.

![Figure 2: CR's operation when it receives (a) an Interest (b) a Data](image)

When an Interest arrives (arrow 1), CR checks its database in the order: CS, PIT, and FIB. If no matching Data is found in CS, the CR checks its PIT. If a longest-prefix matching entry is found and the Interest's nonce value is not recorded yet, the Interest's incoming interface will be added to the matching PIT entry (in particular, to its list of incoming interfaces). The Interest will be discarded because there is already an Interest for this Data sent upstream.

In case there is no matching PIT entry, CR checks its FIB for a longest-prefix match, in order to find an outgoing interface. If a matching entry is found, the incoming interface is removed from the FIB entry (obviously, the Interest and the requested Data cannot come from a same interface. Doing this will reduce unnecessary traffic and eliminate loops). Then, the Interest is duplicated and forwarded via all of the remaining interfaces (arrow 2) of the matching entry. A new PIT entry is also created for this Interest, in order to forward a Data packet later. Gradually, the Interest will arrive to a content provider (arrow 3). In the worst case, if no matching entry in FIB is found, the decision will be left to the forwarding strategy module.
When a Data arrives, the CR performs exact-matching on its CS first (arrow 4). If there is a CS match, the Data packet is duplicated, so it is discarded. If there is no CS match but a PIT longest-prefix match is found, this Data responds for an Interest sent by this CR. The Data is cached in CS and its incoming interface is also eliminated in the matching PIT entry (since Interest and Data do not come from the same interface). The Data packet is then forwarded via all remaining interfaces in the matching PIT entry (arrow 5). After forwarding Data packet, the matching PIT entry of this Data packet is erased immediately, implying that “One Interest retrieves at most one Data packet”. PIT entries that do not receive a matching Data after a period of time will be expired. If there is no PIT match, CR concludes that the Data packet is unsolicited and then discards it. Eventually, the Data packet arrives the requester (arrow 6).

If a matching cached copy is found in CS when an Interest arrives at the router (arrow 7), CR immediately forwards that cached copy to the incoming interface of Interest (arrow 8). In case there are many matching cached copies found, the selector field of the Interest will take part in the selection. Then, the Interest will be discarded.

### 2.4.1 Forwarding Strategy Module

Beside the FIB table, a content router also relies on a forwarding strategy module to make its decisions in packet forwarding process. These decisions include: (1) which Interests should be forwarded to which matching interfaces; (2) how many unsatisfied Interests should be allowed in the PIT; (3) handling Interests with different priority; (4) balancing the amount of forwarded Interests when there are many outgoing interfaces found and (5) finding other paths to forward Interests.
when a failure is acknowledged. An Interest cannot be satisfied for many reasons, e.g. the upstream link goes down; no matching entry in FIB or a congestion is occurring.

2.5 NDN Transport

NDN does not have a separate transport layer. Functions of today’s transport layer protocols, such as multiplexing, segmenting, reliable delivery, congestion control, can be handled by other features of NDN. Multiplexing/de-multiplexing among application processes; segmenting/reassembling segments of data can be done by using hierarchical names. The segment number is already solved in the name structure. Packets of different application processes can be distinguished by using one component of the content name as an identifier.

NDN is designed so that it can operate on top of unreliable packet delivery services. In order to ensure the reliability of delivery, unsatisfied Interests will expire after a period of time and the router will emit a NACK packet to the downstream node. If the requester still wants to get the data, it has to retransmit Interest for that data or find other paths to forward the Interest. In addition, NDN’s forwarding mechanism (section 2.4) helps to reduce significant duplicated Data and Interests. Each Interest has a nonce field in order to detect and to eliminate duplicate Interest (section 2.2). Besides, checking PIT entry before forwarding an Interest helps eliminate redundant Interests that request for the same content. Checking CS before forwarding a Data, in the same way, helps eliminate duplicate Data traffic.

Flow control and congestion control no longer depend on end hosts. By keeping many outgoing interfaces in a FIB entry, Interests can be satisfied by many providers, reducing the load to a particular provider. Flow balance is also maintained in the network by the basic rule in forwarding "One Interest retrieves at most one Data packet". CR can manage traffic load through PIT: when it is overloaded by Data traffic of a specific interface, it can stop sending Interest to that interface. Caches also help data retransmission during congestion. If there is a CR who already caches a Data packet before it is dropped at a congestion link, the requester can retransmit another Interest and it will be satisfied by the cached copy in the CR, thus reducing the latency during congestion.

Each FIB-entry and PIT-entry can have more than one interface. Thus, NDN can support multipath and provide a rich connectivity for nodes in its network.

2.6 Basic Security Mechanisms

Signed Info and Signature in Data packet provide an intrinsic mechanism for authenticity and integrity. The computation process of Signature has been described in section 2.2. When a requester receives a Data packet, nodes can retrieve publisher’s public key with supporting information of Signed info. The key is then used to decrypt the Signature. The result is then compared to the hash value of Signed Info, Content Name and Content fields. This process allows any node to verify data-integrity of the Data packet received and authenticity of the publisher.
3 Approaches of Virtualization

IBM first developed in 1960’s the virtual machine (VM) concept to enable time-sharing and resource-sharing on a very expensive mainframe computer. The main motivation was to reduce the hardware investment cost while improving productivity because virtual machines was designed to allow the same computer to be shared among a large group of users as if it were several machines.

Since this first VM design by IBM, techniques and use cases have evolved, but the principle of virtualization remains the same, which consists of dividing the resources of a single physical computer into multiple isolated and independent execution environments (operating systems or applications), also named as Virtual Machines (VMs) or Virtual Environments (VEs). Based on the Popek and Goldberg requirements [6] that describe guidelines that the Instruction Set Architecture (ISA) of a physical machine needs to respect, virtualization creates an environment as if it is working on a unique server. Through a Virtual Machine Monitor (VMM), which is a piece of software providing the abstraction of underlying hardware, several VMs can be concurrently executed on a single physical platform whose resources are better utilized (resources initially allocated to a VM by the VMM can be dynamically or statically reconfigured when needed). Each VM provides a sandbox that isolates the guest system environment from other guest environments. Thus a failure on a VM does not affect other VMs on the same host machine.

After drawing up the important definitions and notations commonly used in virtualization, we provide a detailed overview of existing virtualization techniques. We then analyze how virtualization is nowadays used in data centers for more flexibility and agility. Finally we describe the open-source software OpenStack architecture for deploying a virtualized infrastructure over a large pool of compute, storage, and networking resources throughout a data center.

3.1 Definitions

The following table lists all the terms that are generally used in virtualization. Some words may have different meanings in literature, so we specify in the document the definitions we use for the present document and during all the project lifetime.

<table>
<thead>
<tr>
<th>Wording</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Host Operating System (OS), host hardware</td>
<td>The host machine is the actual machine, i.e. host hardware on which the virtualization takes place. The operating system on this host machine is called the host OS.</td>
</tr>
<tr>
<td>Guest Operating System (OS), guest applications, et</td>
<td>The guest operating system is the operating system installed inside a virtual machine.</td>
</tr>
<tr>
<td>Virtual Machine (VM)</td>
<td>A virtual machine (VM) is a software-based partition on a physical server using a software layer that abstracts the underlying hardware to mimic a real machine, capable of hosting a guest operating system and hence performing tasks such as running applications and programs like a separate computer.</td>
</tr>
<tr>
<td>Virtual Machine Monitor – VMM (type-2 virtualization)</td>
<td>Software component running on the host operating system (on a physical machine, i.e. the host machine) that hosts virtual machines and abstracts the underlying physical resources for use by those guest virtual machines.</td>
</tr>
<tr>
<td>Hypervisor (type-1 virtualization)</td>
<td>Virtualization software that runs directly on the host hardware, contrary to the VMM that runs on a host operating system to provide virtualization services.</td>
</tr>
<tr>
<td>Container</td>
<td>Sandbox provided by the kernel of the host operating system to isolate an application. The kernel manages memory and filesystem access the same way as if the application were running on the host system. Containers obtain access to resources from the host over normal userland facilities, so they appear as normal processes on the host system.</td>
</tr>
<tr>
<td>Partitions</td>
<td>Multiple VMs run on one physical machine at the same time.</td>
</tr>
<tr>
<td>------------</td>
<td>----------------------------------------------------------</td>
</tr>
<tr>
<td>Isomorphism</td>
<td>Virtualization constructs an isomorphism, i.e. one-to-one mapping, from guest to host. For any operation that modifies the guest state, there is a corresponding sequence in the host system that performs an equivalent modification of the host state.</td>
</tr>
<tr>
<td>Isolation</td>
<td>VMs are isolated, i.e. are full protected and safely separated, hence they do not affect each other. Isolation is provided i) either by a Virtual Machine Monitor (VMM), a piece of software that abstracts host hardware and acts as a bridge to translate guest OS instructions into access requests to hardware resources; ii) or by the kernel of the host OS in the case of container-based virtualization.</td>
</tr>
<tr>
<td>Encapsulation</td>
<td>All VM state can be saved in a file format (i.e., you can operate on a VM as if you operate on a file – e.g. copy, delete, move).</td>
</tr>
<tr>
<td>Interposition</td>
<td>All actions in a VM go through virtualizing software (i.e. VMM or hypervisor), which can inspect, modify, and deny operations.</td>
</tr>
<tr>
<td>Hardware-independence</td>
<td>VMs can be run as is after migration to different hardware platforms.</td>
</tr>
<tr>
<td>Emulation</td>
<td>Emulation and virtualization are two distinct functions. Emulation means that if we want a piece of software (e.g. an old video game), initially developed for System A (e.g. an obsolete platform), to run on another System B (e.g. a new modern platform), we need to make System B “emulate” the working of System A. The software then runs on an emulation of System A. When a device is being emulated, a software-based construct is used to replace a hardware component. With emulation, an entire machine can then be created/reproduced. Virtualization is a technique for using computing resources and devices differently and more efficiently. This involves using a software layer (the VMM) for hardware abstraction and control to isolate a partition (i.e. the virtual machine) on the host server with its own software-based CPU, RAM, hard disk and network connection.</td>
</tr>
<tr>
<td>Snapshot</td>
<td>A snapshot is a file-based representation of the state of a VM: it contains an image of the VMs disk, RAM, and devices at the time it was taken. The functionality can be used for features like performing image level backups of the VMs without ever shutting them down.</td>
</tr>
<tr>
<td>Migration</td>
<td>Moving a VM from one host server to another.</td>
</tr>
</tbody>
</table>

**Table 1: Definitions used in virtualization**

### 3.2 Types of Virtualization Technology

Virtualization is a technology that introduces a layer of abstraction between computing, storage and networking hardware, and the applications running on it. This means that the underlying physical resources (CPU, memory, disk and network) are shared, and there can be multiple systems (or virtual machines) running on a single piece of hardware simultaneously and concurrently.
The virtualizing layer of hardware abstraction then provides the following features (defined in Table 1): partitioning, isomorphism, isolation, encapsulation, interposition and hardware-independence.

There is no common agreement for naming the existing virtualization techniques as a same technique can be named differently in literature. The techniques mostly differ in the degree of host hardware abstraction and the methods used for virtualization (Figure 4).

Thus, on the basis of their abstraction levels, we can highlight, as illustrated in Figure 4, three main types of virtualization:

- **User-mode hosted virtualization** (also called as type-2 virtualization): the entire hardware architecture is replicated virtually through a layer of software run on the host operating system. This layer of software, named Virtual Machine Monitor (VMM), hosts one or several virtual machines that can execute unmodified operating systems or applications over the virtual hardware as if they were on the real hardware of the host server.
- **Native/bare-metal virtualization** (also called as type-1 virtualization or hypervisor virtualization): virtualizing software runs directly on the host hardware to control the hardware, without the need of a host OS.
- **OS-level virtualization** (also called as type-0 virtualization or container-based virtualization): this is not really virtualization, but based on the host OS kernel which provides multiple user space instances, all securely isolated from each other. These user space instances, known as containers, allow users within the container to experience operations as if they are working on their own dedicated server.

The Figure 5 represents the evolution of virtualization technologies and shows the trends towards lower levels of hardware abstraction.

The following sections describe the different types of virtualization techniques, along with advantages and disadvantages of each approach, as well as some relevant references to existing performance evaluation reports in the literature, which can be useful for the project.
3.2.1 User-Mode Hosted Virtualization (Type-2 Virtualization)

3.2.1.1 Principle

In user-mode hosted virtualization (Figure 6), a piece of virtualization software, referred to as the Virtual Machine Monitor (VMM), is executed, as any other application, on the host operating system (such as Windows, Linux/Unix, or Mac OS) of a physical machine to provide virtualization services. Within the VMM, virtual machines (VMs) are created to run the guest operating systems on the host machine independently and with a safe isolation so that failures in one VM cannot affect another. The VMM actually handles an abstraction of hardware resources (processor, I/O devices, storage, memory, network access) for the virtual machines, ensuring they cannot disrupt each other, in addition to providing control interfaces for higher level administration and monitoring tools (the control flow in Figure 6 for an operator to manage the VMs). As such, the VMM provides a mapping of its interface and all virtual resources visible through that interface, to the interface and resources of the real hardware. This data flow, as illustrated by the red arrow-directed path in Figure 6, allows an access control and a mapping of guest-level privileged instructions toward underlying hardware resources. Through the transparent isomorphism between real and virtual hardware, guest operating systems and software can then run without modifications on the virtual hardware and behave exactly as they would on the original hardware. Each guest operating system then appears to have the host’s processor, memory, and other resources all to itself.

The role of VMM to Guest OS in a virtualized environment is similar to the role of OS to user space programs in a non-virtualized environment. The VMM is then also responsible for managing virtual machines (e.g. start, stop, snapshots, migration, etc. – the control flow in Figure 6).

Some examples of user-mode virtualization technologies include VMware Server, VirtualBox, Microsoft Virtual PC and Parallels. The article [7] presents a performance comparison (CPU, disk access) for two free workstation virtualization tools: Oracle VirtualBox and Microsoft Virtual PC.
User-mode hosted virtualization is also called type-2 virtualization to make a difference with the hypervisor-based (or type-1) virtualization techniques that have emerged later and can have a more direct access to the host hardware.

### 3.2.1.2 Advantages

The hardware emulation provided by the VMM in user-mode virtualization allows to run any arbitrary operating system without modifications on the virtual hardware: guest OS is not aware that it is not running on real hardware, and no special CPU hardware virtualization support is required, but this is also a drawback as we describe in the next subsection.

### 3.2.1.3 Drawbacks

In user-mode virtualization, as VM are hosted within the software VMM that behaves as any other application run in the user space of the host OS, CPU instructions that require additional privileges may not be executed in user space. The VMM needs then to analyse the instructions of the executed code from the guest system, and then replace any privileged instructions with safe emulations on the fly. In addition, when also considering the multiple layers of abstraction between the guest operating systems and the underlying host hardware, type-2 virtualization does not contribute to provide high levels of virtual machine performance.

### 3.2.2 Native or Bare-Metal Virtualization (Type-1 Virtualization)

#### 3.2.2.1 Principle

The x86 computer architecture provides the CPU with different operating modes to execute instructions with different privilege levels (see Figure 7). Those modes are organized using hierarchical protection rings from most privileged (i.e. most trusted, usually ring 0) to least privileged (i.e. least trusted, usually ring 3). This design allows the operating system to run with more privileges than application software. Ring 0 corresponds to the kernel mode or supervisor mode, i.e. the level with the most privileges, where the operating system kernel runs and interacts most directly with the physical hardware such as the CPU and memory. All other code such as applications running on
the operating system operates in less privileged rings, typically ring 3 (also known as the user mode).

![Diagram showing privileged levels in the x86 computer architecture]

**Figure 7: Privileged levels in the x86 computer architecture**

Native virtualization is based on a software component, referred to as a hypervisor, which is executed directly on the hardware of the host server in ring 0 to control the hardware and to manage and monitor guest operating systems. As illustrated in Figure 8, the main feature of this architecture is the lack of an existing OS on the host machine: the hypervisor sits directly on top of the hardware, hence the term "bare-metal virtualization".

![Diagram showing native virtualization (type-1 virtualization)]

**Figure 8: Native virtualization (type-1 virtualization)**

The term “hypervisor” is often used mistakenly to also refer to the VMM in user-mode hosted virtualization. Logically, native virtualization takes place in ring 0 of the CPU and controls guest OS that also expect to run the kernel within the supervisor mode in ring 0, hence the name “hypervisor” for the native virtualization software. However, in practice, the kernel of any guest operating system running on the hypervisor must run in less privileged CPU rings because the hypervisor is already installed in ring 0 directly on the host hardware and guest OS should then behave as less privileged “applications” on the hypervisor similarly to the role of an OS to user space programs in a non-virtualized environment. To this end, different solutions have been proposed, as described hereafter, for handling sensitive and privileged instructions to virtualize the CPU on the x86 architecture.
3.2.2.1.1 Paravirtualization

Using paravirtualization (also known as OS-assisted virtualization – see Figure 9), the kernel of the guest operating system must be explicitly pre-processed (at build-time, load-time, etc.) to run on the hypervisor. This involves replacing direct invocation of privileged instructions (that would normally run in ring 0 of the CPU) with explicit calls to hypervisor APIs (known as hypercalls). The hypervisor in turn performs the task on behalf of the kernel of the guest OS. Only open source operating systems such as Linux are generally adapted to target a specific hypervisor. For proprietary operating systems, the owners shall be responsible for modifying the kernel code, but they can refuse to do so for strategic reasons. The paravirtualization approach is used by products such as Xen and UML.

Figure 9: Paravirtualization

With additional channels of direct communication between the hypervisors and the guest operating systems, without the overhead imposed by the emulation of the system's resources, paravirtualization provides greater performance and efficiency than other virtualization approaches. In particular, this enables near-native performance for disk and network operations. In return, however, the drawbacks of paravirtualization is that its compatibility and portability are poor, because we must tune the guest OS to translate privileged instructions in the guest kernel into specific calls to the hypervisor. In a way, the guest OS is aware that it is being virtualized.

3.2.2.1.2 Full Virtualization

Full virtualization provides support for unmodified guest operating systems and mainly consists in performing ring deprivleging, which involves running the guest OS at a ring higher than 0 (see Figure 10). The guest OS is fully abstracted from the underlying hardware by the hypervisor. The guest OS is not aware it is being virtualized and its kernel does not need to be modified. Indeed, the hypervisor simulates a complete hardware environment and provides CPU emulation to handle and translate all privileged operations made by the guest operating system. VMware, for instance, uses a technique called binary translation to automatically modify sensitive instructions on the fly with new sequences of safe instructions on the virtual hardware (i.e. rewrite in terms of ring 3 instructions certain ring 0 instructions from the guest kernel); results are also cached for future use.

VMware ESXi and Microsoft Virtual Server are examples of type-1 hypervisor providing full virtualisation.
The interception and translation of privileged operations in full virtualization requires both time and system resources to operate resulting in inferior performance levels when compared to those provided by paravirtualization.

### 3.2.2.1.3 Hardware-assisted Virtualization

Hardware-assisted virtualization relies on hardware capabilities, mainly the host processors, and was added to the x86 architecture in 2006 with the arrival of AMD-V and Intel VT-x processor additions. These new processor technologies provide a new CPU execution mode under the traditional ring 0, so that the hypervisor can operate in this new lower root execution mode, leaving ring 0 available for unmodified guest operating systems (see Figure 11). With greater hardware support, nowadays, the latest processor models allows for substantial speed improvements.
Citrix XenServer and Microsoft's Hyper-V are examples of type-1 hypervisor with hardware-assisted virtualization. VMWare ESXi is also able to use hardware-assisted virtualisation for the CPU as well as for the Memory Management Unit (MMU).

The paper [8] compares the performances of VMware ESXi 5, Citrix Xen Server 6.0.2 and Hyper-V, all type-1 hardware assisted hypervisors.

### 3.2.2.2 Advantages

Since they run directly on bare hardware (physical servers), hypervisor-based (type-1) virtualization provides virtualized environments with little overhead and is more efficient than hosted architectures (type-2 virtualization), enabling thus greater scalability, robustness and performance.

Paravirtualization does not require virtualization extensions from the host CPU and thus enables virtualization on hardware architectures that do not support hardware-assisted virtualization.

Full system virtualization has the benefit that guest operating systems and applications can run on it unmodified, completely oblivious to the hardware environment in which they are actually running. This allows to simplify migration and portability as the same guest OS instance can run on another virtualized or native hardware. Full virtualization is also the only option that does not need hardware or OS assistance to virtualize protected and privileged instructions.

### 3.2.2.3 Drawbacks

Paravirtualization involves modification of the guest OS to run on a specific hypervisor, resulting in VMs that suffer from lack of compatibility and are not very portable on another hardware. While full virtualization supports unmodified guest OS, translation of privileged instructions from guest OS introduces higher degree of performance overhead and is a real challenge that requires complex combination of hardware and software techniques because some sensitive instructions cannot effectively be virtualized as they have different semantics when they are not run in ring 0 of the CPU.

### 3.2.3 OS-Level Virtualization (Type-0 Virtualization)

#### 3.2.3.1 Principle

Operating-system-level virtualization (Figure 12) does not really virtualize the host hardware, but the kernel of the host OS isolates guest users by providing multiple isolated user space instances, instead of just one, named as containers. The container concept is somewhat equivalent to the chroot mechanism in Unix-like systems, and as such, containers first took an important role with FreeBSD Jails, built on chroot and available since FreeBSD 4.x for enhancing security.

Nowadays, container-based partitions continue to be enhanced in their usefulness, performance, reliability, and security. ‘Modern’ containers such as Docker, LXC (Linux Containers) or lmctfy (Let Me Contain That For You) rely now on the cgroups functionality of the Linux kernel.

Container-based technologies need a patched kernel to run the virtual environments. The kernel then provides isolation and performs resource management, so that all the containers effectively have their own file system, processes, memory, devices, etc. We have one container for each virtual environment.
Figure 12: OS-Level Virtualization, a.k.a. Containers Virtualization (type-0 virtualization)

Docker, LXC, Imctfy, OpenVZ, Virtuozzo, Linux-VServer, Solaris Zones and FreeBSD Jails are examples of OS-level virtualization. Note that Sandboxie and iCore Virtual Accounts are containers for Windows hosts.

In paper [9] a performance comparison between traditional VMs and Linux containers will be found. The authors compare the type-1 hypervisor KVM to the Docker Linux container. They show how different isolation and resource control techniques induce differences in performance. The idea of paper [10] is to compare the performance of three hypervisors: one supporting paravirtualization (Xen-PV), other supporting container virtualization (OpenVZ) and the last one supporting full virtualization (XenServer).

3.2.3.2 Advantages

Container technology is a lightweight kind of virtualization. The main difference between containers and 'regular' virtual machines is that there is no overhead: everything takes place directly within the host's kernel. This means that contrary to VMs, applications in a container use the system call interface of the host OS to access hardware resources and do not need to rely on hardware emulation or be run in an intermediate virtualization layer.

Type-2 and type-1 virtualization usually have limits in terms of how many CPUs and memory a guest can address, whereas the container-based solutions should be able to address as many CPUs and as much RAM as the host kernel. This is useful when we need to consolidate a large number of Linux instances.

3.2.3.3 Drawbacks

With container-based virtualization, installing a guest OS is however not as straightforward. The only relationship between the host OS and the container is the host kernel. This means that we cannot install a different guest kernel than the one used in the host OS and we can only run Linux based distribution/binaries within the container on a Linux host (however we are not limited to a specific distribution).
3.3 Vertical and Horizontal Virtualization

Nowadays, virtualization techniques are used for the abstraction of the physical hardware, with the objectives of i) vertically scaling resources to simplify management and operational tasks (e.g. reducing equipment and energy costs, easier backups, better disaster recovery, simpler server migration, better testing new applications, etc.); ii) horizontally scaling services to create clusters of multiple logical resources for storage, processing, or networking, and then propose those resources to users as cloud services.

Vertical virtualization means multiple VMs can be deployed on a single host server and presents the following advantages:

- **Server consolidation** reduces hardware and operating costs and energy costs; reduce the overall footprint of the data center, i.e. far fewer servers, less networking gear, smaller number of racks needed, resulting in smaller surface required for the data center.
- **Security and reliability of applications** i.e. separating and isolating applications that require different security levels.
- **Create a virtual machine for each application** e.g. either for testing purposes in an isolated lab network, or reducing server waste by provisioning virtual machines with the exact amount of hardware resources that it needs.
- **Disaster recovery**: just backup VMs or replicate them to another site, then restart VMs on a different host.

As the hardware resources on a host server are limited, using multiple physical machine instances or combining resources (CPU, storage, memory, network) from a distributed cluster of machines allows achieving horizontal virtualization. This amounts to clustering multiple hardware entities so that they work as a single logical unit on which we can perform virtualization. An important advantage of horizontal scalability is that it can provide the ability to increase capacity and performance on the fly by simply adding new hardware entities to the cluster. Another advantage is to allow an operator to prepare the migration towards cloud services, e.g. for offering on-demand computing resources or acting as IaaS provider, by provisioning and managing large networks of virtual machines. The open-source cloud operating system OpenStack provides thus all the facilities for horizontally scaling virtualized applications or services through a large pool of compute, storage, and networking resources.

![Figure 13: Horizontal virtualization: virtualization across distributed back-end resources](image-url)
3.4 Virtualization in Data Centers

Data centers are IT infrastructures that gather clusters of physical servers hosting several virtual machines. As such, they are typical examples of a mix of vertical and horizontal virtualization solutions. However, in order to provide a high performance level and to deal with multi-tenancy, they not only require the virtualization of applications and operating systems (i.e. virtualization of computation) as explained below, but they also require the virtualization of other components, such as the network access, and storage. Indeed, regarding the network perspective, the equipment forming the network like the routers or the firewalls may either be physical or virtual, all of them having an access to one or several IP networks. This results in a logical architecture like the one shown in Figure 14.

![Figure 14: A logical architecture composed of physical and virtual entities](image)

To set up such an infrastructure, the network card access should be virtualized as well as the hardware components (computation, memory, storage, etc.).

3.4.1 Network Interface Virtualization

According to the requirements of a tenant hosting virtual machines in a data-center as well as those of the infrastructure operator, one can distinguish several classes of virtualization solutions for the network access. We illustrate them with a simple example: two workstations virtualized on a physical workstation (Figure 15).

![Figure 15: Two VMs on a physical workstation](image)
Network virtualization comes with the need to give network access to each of the virtualized workstations though the unique host workstation physical Network Interface Card (NIC). This is a classical networking situation that may be solved using: (a) a switch (Figure 16-a), (b) a Virtual Local Area Network (VLAN) capable switch (Figure 16-b), (c) a switch-router (Figure 16-c), or (d) a switch-router performing Network Address Translation (NAT) (Figure 16-d).

Each class of solution presented in the figure above, leads to different features that may be compared to each other using several criteria (Table 2). First of all is the location in the logical architecture it'll be possible to place the produced VMs in (“VM location” in the table). The second criterion is the IP network the VMs will be able to belong to (“VM IP network”). The third criterion is the layer-two protocol that the host machine will have to use (“Host L2 protocol”). It is then interesting to have a look on the kind of access the host machine may dispose of for its needs (“Network host access”). The fifth criterion deals with the choice of network functions the hypervisor will play induces the location where the host machine has to be plugged on the physical architecture (“Host machine position”). Finally, the four solutions presented in Figure 16 differ in their aim. Indeed, they make it possible to extend physical network architecture, but the added virtual network function differs from one solution to another. We highlight it in the sixth criterion (“VNF role”).

<table>
<thead>
<tr>
<th>Type of solution</th>
<th>VM location</th>
<th>VM IP network</th>
<th>Host L2 protocol</th>
<th>Network host access</th>
<th>Host machine position</th>
<th>VNF role</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 16 (a)</td>
<td>In the host machine communication shared medium</td>
<td>In the host machine IP network</td>
<td>Ethernet, with one MAC address per VM, and an other one for its needs</td>
<td>Access Network on which it is physically plugged</td>
<td>In a access network</td>
<td>Extension of an access network</td>
</tr>
<tr>
<td>Figure 16 (b)</td>
<td>In a L2 backbone</td>
<td>Different IP networks already existing in the physical architecture</td>
<td>802.1q</td>
<td>One of the networks transported in the L2 backbone</td>
<td>Plugged on a L2 backbone</td>
<td>Extension of a L2 backbone</td>
</tr>
<tr>
<td>Figure 16 (c)</td>
<td>In a media not already existing</td>
<td>In an IP networks not already used</td>
<td>Ethernet, as a router does</td>
<td>In each of the virtual networks created, or in the interconnection to the physical architecture</td>
<td>Plugged on a L3 backbone</td>
<td>Extension of a L3 backbone, with participation to the routing protocol</td>
</tr>
<tr>
<td>Figure 16 (d)</td>
<td>In a media kept invisible to the rest of architecture</td>
<td>Any IP networks</td>
<td>Ethernet</td>
<td>Access Network on which it physically plugged</td>
<td>In an access network</td>
<td>No role (not visible)</td>
</tr>
</tbody>
</table>

Table 2: Features of network virtualization solution classes

The most used solution is this last one (switch-router performing NAT) first because it uses only Ethernet protocol, and there is thus no need for a special NIC on the host workstation. Moreover, there is no need to worry about VMs IP network choice. This solution is very suitable for client virtualization. In case of server virtualization, static NAT (also referred as reverse NAT) should be implemented in the virtualized networking component.
3.4.2 Virtualized Components

If the previous section highlighted the different aims of the network access virtualization, their implementation may be addressed through any mix of physical and virtualized components (fully virtualized switches, virtualized extensions of physical switches, fully virtualized routers, firewalls, etc.). Beyond, any other kinds of resources a data center infrastructure has to provide may be subjected to such physical implementation of components while being offered to tenants by leveraging virtualization. These are computation resources, processing memory, storage memory, and eventually some special data center designed switching equipment. This set of components is depicted in Figure 17.

Figure 17: The operator data center structure

First of all, a data center infrastructure relies on a centralized management model where, all components are managed and configured by a dedicated server. Computation and processing memory are provisioned with suitable servers. For example, these are the compute nodes in the OpenStack architecture or the host nodes in the VMware architecture. These servers have to be connected together to enable special data center actions like migrations of VMs from an execution server to another one without to stop them or dynamic resource allocation in order to optimize energy consumption.

Storage capabilities are provisioned with a group of storage servers, generally referred as SAN for Storage Area Network. This is useful to store the actual representation of an existing VM, either alive or shutdown, including its hard disk to which rapid access is to be guaranteed and also previous representations of VMs called « snapshots ». The Logical Unit Numbers (LUNs) created in a SAN are equivalent to local disks for machines being given access to these LUNs although they do not reside on the same physical machine. In a data center, accesses to LUNs are given to VMs execution hosts in order to let VMs operating systems write in block mode in what they believe to be their local hard disk, and also to let the data center virtualization software write files representing the VMs.

As shown in Figure 17, SAN and VMs execution hosts are linked together using switching equipment supporting specially designed high quality network. Most used network solution is Fibre Channel. A combination of Ethernet and Fibre Channel is also commonly used and is known as Fibre Channel over Ethernet (FCoE). A lower performance level solution combines Ethernet, IP, TCP and iSCSI protocols. Between hosts, the link is useful to transport alive VMs migration flows. Inter VMs flows are generally separated from migration flows and transit through external to data center switch.
Special purpose designed hosts are used specially to provide client architectures with network elements like routers, firewalls, deep packet inspection tools, and so on (Neutron in OpenStack, and NSX VMware are examples). The aim is here to let the client take network elements « on the shelf », through a web interface (e.g. dashboard), and add them to his architecture (virtualized in the operator data center). These network elements are then provisioned in the data center architecture to meet the client quality of service demand.

3.5 OpenStack Architecture

The previous section describes the overall set of components a data centers has to implement and offer to its clients through virtualization. As an infrastructure example of these components, we consider the case of OpenStack which is the main example of an open source solution, massively used to control and manage a large pool of cloud resources, such as compute, storage, and networking, thus enabling the provision of large-scale data-center services. OpenStack proposes a modular architecture based on multiple environments tools, thus allowing the implementation of heterogeneous architectures as needed.

![Minimal Architecture Example - Service Layout](image)

**Figure 18: Minimal architecture example with OpenStack Networking - Service layout [11]**

As shown in Figure 18, the physical architecture of OpenStack is composed of multiple nodes (Compute, Controller, Network and Storage) whose goal is given below:

- **Controller**: the controller node is the central node for the management of OpenStack services. It manages the databases, message queue services, authentication and authorization for identity management, image-management services and it hosts the administrator dashboard.
- **Compute**: the compute node hosts the cloud instances and as such provides the necessary resources such as processing, memory, network and storage to run them. The recommended hypervisors are currently KVM and XenServer. However, OpenStack can also support other hypervisors such as LXC, QEMU, VMware ESX/ESXi, Xen, Hyper-V and some work has been engaged for the support of Docker.
• Network: OpenStack provides a rich networking environment based on multiple control API for which enable the implementation and control of both tenant based virtual networks as well as the cloud operator underlying infrastructure. The reach of potentially millions of addresses are supported by the set of network components.

• Storage: storage nodes in OpenStack, which are not mandatory for a basic architecture, are devoted to two types of content. The object storage provides a distributed storage system for static data such as virtual machine images, image storage, email storage, backups and archives. By contrast, the bloc storage provides persistent block storage devices to be used as device storage for compute instances.

3.5.1 OpenStack Networking

As described above, a standard OpenStack architecture contains a controller node, a network node, and a number of compute nodes (hypervisors) for hosting virtual machines. Figure 19 shows a typical example of basic OpenStack networking infrastructure that connects the physical nodes. In the latter, we can identify different networks as follows:

• Management network: The management network provides internal communication between OpenStack hosts. This network should only be reachable from inside the cloud (controller, compute, network, and storage).

• Data network: The data network provides the virtual instance with communications capabilities within the cloud infrastructure.

• External network: The external network provides the instances with an access to the Internet.

• API network: The API network may be the same as the external network. It exposes all OpenStack APIs, including the OpenStack Networking API, to tenants. The IP addresses on this network should be reachable by anyone on the Internet.

![Figure 19: Physical network connectivity](image-url)

As shown on Figure 20, within this architecture, one can distinguish the cloud service provider networking part and the tenant one. If they stand for different parts of the whole virtual and logical infrastructure, they all are mapped onto the physical network. The provider networks are those that enables the global communication between any component of the OpenStack infrastructure. For instance, they enable the communication between instances located on different hosts but belonging to the same tenant or the access of the latter to the Internet, whatever the VM migration policies are. By contrast, the tenant networks are those created and managed by users within the tenant's
virtual infrastructure. These networks are fully isolated so that to enable a full independence from other tenants. However, it remains possible to route between tenant networks using GRE and the L3 agent.

![Diagram of tenant and provider networks](image)

Figure 20: Example of tenant and provider networks [12]

### 3.5.2 Virtual Networking Components and Namespaces

In order to implement the networking capabilities identified above, OpenStack relies on several software components that we briefly review below:

- **Open vSwitch**: Open vSwitch ensures the L2 connectivity of OpenStack networks. It can use two different technologies to enable the transport of frames over mutualized links, namely GRE or VXLANs. For example, in the case where two instances belonging to the same tenant are execute on separate hosts, they all will be connected to the Open vSwitch agent running on their respective local host. Then to enable a communication between these two instances, the two Open vSwitch components will encapsulate data into a VXLAN or GRE tunnels.
- **DHCP agent**: The DHCP agent allocates IP addresses to instances from the specified subnet.
- **L3 agent**: The L3 level is needed by OpenStack administrators and tenants to create virtual routers. To that aim, Neutron offers an API extension known as L3-agent which leverages Linux IP stack and iptables to perform L3 forwarding and NAT. It uses Linux network namespaces to provide isolated forwarding contexts.

In order to implement concurrent networking functions over the same physical host and operating system, OpenStack relies on Linux namespaces. The separation of networks into namespaces allows avoiding collision between the physical and the logical networks as well as those between logical networks. Open vSwitch components hosted in both compute and network nodes, as well as L3 agents hosted on the network node rely on Linux namespace to leverage the multi-tenancy. The purpose of network namespaces is to separate network domains and especially networking stacks into separate and independent ones. As such, each tenant network is completely separated from other tenants. Based on this separation, the service providers offer on-demand networking resources whatever the configuration of its backbone infrastructure and those already used by other tenants, thus preventing any collision due to, for instance, concurrent IP addresses spaces.

To conclude, if OpenStack already provides virtual network components as those identified above, the latter are not intended to operate in a NFV compliant way since these components are intended to operate in a data-center virtualization context (vs. a telco-operator infrastructure). Thus, they are only designed and implemented as means that bring communication facilities to VM against value-added networking functions such as those that designate VNF.
4 Towards Network Function Virtualization

As virtualization gained popularity and proved it efficiency in data centers (cost reduction, faster service deployment, greater flexibility, etc.), new use cases are envisioned. We notably observe in recent years the trend towards IT and Telecom convergence, meaning that Telecom operators have looked for getting the same benefits as IT organizations by reusing IT virtualization techniques to virtualize network functions or components into building blocks treated as virtual machines that may be chained (i.e. connected) to create end-to-end communication services. This brings us to the emergence of a new paradigm for Network Function Virtualization (NFV).

4.1 The NFV Architecture and Technical Foundations

4.1.1 The NFV Paradigm

The NFV concept was first introduced in a white paper published in 2012 at the “SDN and OpenFlow World Congress” [13]. This concept is pushed by telecom operators such as AT&T, Verizon, BT, Telefonica, Orange, China Mobile, NTT, etc. Two updated versions of the white paper were published in 2013 [14] and 2014 [15] to add new functionalities and description of the NFV framework. The ETSI NFV ISG has grown since 2012 and is now composed of more than 265 companies (with 34 connectivity providers) and 34 Proof-of-Concepts. The first phase is now over and many documents are available, such as those related to the architectural framework, the terminology, the use cases, the POCs, etc. The second phase of NFV starting in 2015 will be related to ensure interoperability of NFV solutions, interaction of NFV with legacy systems, inter-domain aspects, etc.

NFV is based on the fact that there is now a lot of various equipment in the network, each one deployed for a specific usage, having specificity in the way they run and are managed and thus requiring from the network operational teams specific competencies. Network operators also analyzed that the equipment consume a lot of energy, which is critical in the current costless and green trend.

NFV has then been proposed to offer a most efficient, less consuming and less costly solution. This is done by leveraging IT virtualization techniques for network equipment. This is designed to be valid for all services, but also many different network equipment, being located in data centers or specific locations in the network. Figure 21 presents this objective.

Figure 21: NFV Approach
The expected benefits of the NFV approach are:

- reduced equipment costs and power consumption through consolidating equipment
- increased speed of time to market by minimizing the typical network operator cycle of innovation
- shared resources across services and different customer bases
- services can be rapidly scaled up/down or in/out
- openness of the virtual appliance market to pure software entrants, small players and academia, encouraging more innovation to bring new services and new revenue streams quickly at much lower risk.

4.1.2 The NFV Architectural Framework

The NFV architecture is schematically represented as in Figure 22, with the following building blocks.

![Figure 22: ETSI NFV architecture vision](image)

In an NFV architecture, the elementary building block is the VNF (Virtual Network Function). The VNF is the software developed to be run in the virtualized environment and rendering a given service (or part of a service). VNFs can be chained to form a more complex and final service. Finally VNFs can be deployed over multiple VMs, where each VM hosts a single component of the VNF (i.e., VNFC). However, in some cases, the whole VNF can be deployed in a single VM as well.

The Orchestrator has two main responsibilities [16]: (i) lifecycle management of network services; (ii) orchestration of NFV Infrastructure (NFVI) resources across multiple Virtualized Infrastructure Managers (VIMs).

The VNF Manager (VNFM) is responsible for the lifecycle management of VNF instances. Each VNF instance is assumed to have an associated VNFM. A VNFM may be assigned the management of a single VNF instance or the management of multiple VNF instances of the same type or of different types; most of the VNF Manager functions are assumed to be generic common functions applicable to any type of VNF.

The deployment and operational behaviour of each VNF is captured in a template called Virtualised Network Function Descriptor (VNFD) that is stored in the VNF catalogue. A VNFD is used to create instances of the VNF it represents, and to manage the lifecycle of those instances. A
VNFD has a one-to-one correspondence with a VNF Package, and it fully describes the attributes and requirements necessary to realize such a VNF.

The NFVI (NFV Infrastructure) encompasses all the hardware (e.g., compute, storage, and networking) and software (e.g., hypervisors) components that together provide the infrastructure resources where VNFs are deployed. NFVI resources are assigned to a VNF based on the requirements captured in the VNFD (containing resource allocation criteria, among others), but also taking into consideration specific requirements, constraints, and policies that have been pre-provisioned or are accompanying the request for instantiation and may override certain requirements in the VNFD (e.g., operator policies, geo-location placement, affinity/anti-affinity rules, local regulations).

The VIM is responsible for controlling and managing the NFVI compute, storage and network resources, usually within one operator's infrastructure domain, e.g., NFVI-PoP. A VIM may be specialized in handling a certain type of NFVI resource (e.g., compute-only, storage-only, networking-only), or may be capable of managing multiple types of NFVI resources. Functions performed by the VIM include:

- resource catalogue management,
- orchestrating the allocation/upgrade/release/reclamation of NFVI resources, and managing the association of the virtualized resources to the compute, storage, networking resources,
- supporting the management of VNF forwarding graphs (create, query, update, delete), e.g., by creating and maintaining virtual links, virtual networks, sub-nets, and ports,
- management of the NFVI capacity/inventory of virtualized hardware resources (compute, storage, networking) and software resources (e.g., hypervisors).

The OSS/BSS (Operations/Business Support System) are the combination of the operator's other operations and business support functions that are not otherwise explicitly captured, but are expected to have information exchanges with functional blocks in the NFV architectural framework. OSS/BSS functions may provide management and orchestration of legacy systems and may have full E2E visibility of services provided by legacy network functions in an operator’s network.

### 4.1.3 NFV POCs

The ETSI NFV ISG has developed an NFV PoC Framework to coordinate and promote multi-vendor Proofs of Concept illustrating key aspects of NFV ISG work. The goal is to build awareness and confidence and to encourage the development of an open ecosystem by integrating components from different players. Proofs of Concept also help to develop a diverse, open, NFV ecosystem. Results from PoCs may guide the work in the NFV ISG by providing feedback on interoperability and other technical challenges. There are currently 34 registered POCs (already done or ongoing). The list can be found in: [http://www.etsi.org/technologies-clusters/technologies/nfv/nfv-poc](http://www.etsi.org/technologies-clusters/technologies/nfv/nfv-poc).

Some of them are related to Cloud, others to mobile network (e.g. vEPC or VRAN), or service chaining, other also to SDN. Amongst them, only one is related to the DOCTOR’S objectives (secure data routing in virtualized environment). The “VNF Router Performance with DDoS Functionality” POC (AT&T, Telefonica, Brocade, Intel, Spirent) aims to demonstrate various Layer 2-4 DDoS attacks, e.g. NTP reflection attack, being handled in real-time using a VNF router. This PoC also characterizes the performance impact of implementing Layer 2-4 DDoS attack detection/mitigation schemes in a VNF router. This POC is based on ETSI GS NFV 004 (Network Functions Virtualization (NFV); Virtualization Requirements) describing the need for appropriate security countermeasures to address security vulnerabilities introduced by the virtualization layer. Behavioral security threats such as Distributed Denial of Service (DDoS) attacks are an ongoing problem in today’s Data Centers and are expected to pose even greater challenges to network operators deploying NFV infrastructures. This problem is expected to be most acute in multi-tenant DCs providing Virtual Network Function as a Service. This PoC demonstrates various Layer 2-4 DDoS attacks being detected and mitigated in real-time by a Virtual switch/router (Brocade Vyatta vRouter 5600) which is running on an Intel x86 server as part of an NFV infrastructure. The Virtual switch/router employs
scalable inline line-rate algorithms for automatically recognizing the large flows which are the cause of DDoS attacks; this should ensure that the Virtual switch/router performance is not impacted.

4.1.4 OPNFV

In September 2014, some NFV actors launched a new initiative related to NFV, the Open Platform for NFV Project (OPNFV) [17]. OPNFV is an open source project hosted by the Linux Foundation. OPNFV is very close to ETSI NFV and will follow NFV ISG recommendations. OPNFV is set up in order to foster the implementation and deployment of NFV. It should also ensure high performance and interoperability between components.

The objectives of OPNFV are twofold: firstly, a collaborative development of an open source platform to promote interoperable NFV solutions; and secondly, to help to stimulate existing open source communities to create the software code or hardware necessary to implement NFV solutions based on common industry requirements.

The initial scope of OPNFV will be on building NFV Infrastructure (NFVI), Virtualized Infrastructure Management (VIM), and including application programmable interfaces (APIs) to other NFV elements, which together form the basic infrastructure required for Virtualized Network Functions (VNF) and Management and Network Orchestration (MANO) components.

4.1.5 The OpenStack Effort for NFV

OpenStack is currently the most acknowledged open-source platform for cloud IaaS infrastructures. As a non-proprietary solution, which intrinsically brings the independence from any hardware component, it also acts as a promising candidate for a NFV supporting infrastructure since the latter aims at fully decoupling the hardware and software parts. From the data-center reference architecture where compute nodes host VM while the network node hosts the underlying layer 2 backbone, the shift toward a NFV compliant architecture is straight since it mainly requires compute nodes to host VNF while keeping other components with similar functionalities. However, if from a functional perspective, the mapping seems straight, it requires adaptations to NFV specific operational constraints of some or even all components.

Following this idea, many initiatives refer to OpenStack as a future platform for NFV. For instance, the OPNFV initiative, presented above, proposes to integrate part of existing solutions such as Open-daylight or OpenStack components to build the future NFV open-source implementation. The OpenStack consortium also owns a dedicated working-group (opened late in 2014) entitled telco-working-group [18] whose purpose is to identify and prioritize the requirements, which are needed to deploy, manage, and run telecommunication services on top of OpenStack. This work includes identifying functional gaps, creating blueprints, submitting and reviewing patches to the relevant OpenStack projects and tracking their completion in support of telecommunication services. Currently, several active blueprints have been identified, each requiring changes to existing OpenStack components, and they are also mapped onto ETSI-NFV use-cases so that these efforts can be related. Finally, RedHat and the CloudBand team of Alcatel-Lucent also work at moving OpenStack from a pure data-center infrastructure to a telco-compliant one [19]. To that aim they concentrate their efforts on specific points they consider as under-addressed in current OpenStack proposals. More specifically, these are: Distribution, Networking, Operations, Lifecycle management and data plane optimization. The final goal of this initiative is to design, implement and deliver a full NFV compliant infrastructure based on Red Hat Enterprise Linux OpenStack Platform.

4.1.6 Enabling NFV by Providing High Performance in Virtualized Networks

One of the lock which could prevent NFV from a massive adoption by telco-operators relies in incapability to provide a high performance data-plane. In other word, the key condition to enable the adoption of this technology relies in its capability of still maintain the networking performance level provided by dedicated proprietary hardware for basic networking functions and appliances. Currently, some technologies exist to propose solutions that enable virtual networking components to get a performance, which is comparable with non-virtualized ones. Among them, one can cite SR-IOV (Single Root I/O Virtualization), a technology which proposes a high performance access to physical
networking devices to virtual machines by bypassing the software switch which acts as a throughput bottleneck. If this solution is valuable regarding the performance it offers, it requires network interface controllers (NIC) to support it natively and as a consequence, it does not respect the NFV key concept which consists in bringing a full independence of networking from the hardware layer. To solve this issue some companies, such as 6Wind propose to exclusively use a software acceleration to get, for a telco-operator, the expected networking performance [20]. The 6Wind solution, entitled 6WindGate, exclusively focuses on the NFVI part of the NFV reference architecture and it consists in:

- An architecture independent "Fast Path Modules" which proposes a generic and processor-independent source code optimized regarding the cycle-level and pipeline-level;
- An architecture specific "Fast Path Networking SDK" which enables a zero-level API for upper fast path modules while ensuring the interface with hardware and proprietary technologies (for instance Intel DPDK).

To illustrate the benefits of the 6Wind technology, we summarize here the result of a basic benchmark performed by 6Wind. It consists in a VM performing IPV4/IPV6 forwarding operations (on a RedHat Linux deployed over Dell R720 servers with 12 10Gigabits interfaces). By implementing the 6Wind Gate modules in both the VM and the hypervisors, the layer-two network throughput goes from 7.2 Gbps to 118.4 Gbps which stands for a relevant improvement.

Apart from this basic benchmark, 6Wind advocate the use of its technology in OpenStack and identifies three use-cases, which are data-center virtualization, appliance virtualization and network function virtualization. In the latter case, which stands for the core of the Doctor project, VNF are hosted and executed in compute nodes, while the underlying networking capabilities are provided by Open vSwitch instances executed and tied on both compute and network nodes. In this context, identified performance bottlenecks concern both East-West data transfer (i.e. VNF to VNF for service chaining) and North-South data transfer for any remote communications, mainly due to Linux virtual switching. By integrating the 6Wind Gate in such an infrastructure, 6Wind claims (based on a specific benchmark) that East-West data transfer can reach a 240Gbps throughput while North-South data transfer may reach 720Gbps. Such results clearly state that (1) the 6Wind technology may be a key enabler for the adoption of the NFV paradigm and (2) OpenStack may be one the key supporting infrastructure.

4.2 NFV Hand and Hand with SDN

SDN (Software-Defined Networking) is the name given to a new manner of architecting and deploying networks. The principle features of this vision of future networks are: (i) separating the control and data planes, (ii) centralizing the control plane, (iii) making the network programmable. Figure 23 shows the high level description of a SDN architecture using the OpenFlow protocol.
The upper layer is the application layer, where applications defines their requirements and needs to the control layer in order to configure the network equipment. Those applications can be virtualized, so the application layer can also include the different network functions that can be virtualized following the NFV paradigm, using for example the open-source OpenStack for scaling up over a cluster of hardware resources.

The control layer is the main component of SDN. It is where the SDN controller is. The SDN controller has a complete view of the network and is able to define polices and rules in order to satisfy applications requirements based on the network conditions. The forwarding rules are sent by the controller to the programmable network devices, located in the infrastructure layer. OpenDaylight is an open-source framework providing a software bundle that covers the major common components required in the control layer to build an SDN solution. Moreover, OpenDaylight is well integrated with OpenStack to enable SDN in hand and hand with NFV for networks at any size and scale.

The infrastructure layer is where the physical programmable devices are. They route packets according to the configured rules, sent by the controller.

For the separation of roles and planes, interfaces are defined. The southbound interface (between the infrastructure layer and the control layer) is now well defined, for instance the OpenFlow protocol specifications. The northbound interface (between the application layer and control layer) still requires work to well define the requirements and related network configurations in an abstracted way.

NFV brings agility, flexibility in the network while SDN brings programmability. We can see that both technologies can be very complementary. NFV can help to deploy network functions in virtualized environment and SDN can help to dynamically configure those virtualized network functions.

There are few papers describing this combined use of the two technologies. Amongst the most ones in relation to the Doctor’s objectives, we can mention [21]. This paper introduces the deployment of the Information-Centric Networking (ICN) paradigm with emerging technologies such as Network Function Virtualization (NFV) and Software Defined Networking (SDN). They propose a framework to design an ICN-based service platform as virtualized network functions to enable several edge-cloud services such as enterprise applications, big data analytic, or M2M/IoT services.
This platform is generic to support several ICN protocols and corresponding real-time and non-real-time services leveraging ICN features such as name-based routing, caching, multicasting, and flexible security techniques.

In [22], the authors present the analysis, design, and the first implementation, in a virtualized manner, of the routing function (IPv4 and IPv6). Indeed, they argue that routing is the basic network function offered by network routers and not yet investigated in the NFV community. This paper presents one of the first functional implementations of the NFV concept, through the virtualization of the routing function over an OpenFlow network.

In [23], the authors propose to use NFV to run the mobile network functions as software instances on commodity servers or data centers, while SDN supports a decomposition of the mobile network into control-plane and data-plane functions. The authors argue that taking load and delay into account, there will be areas of the mobile network rather benefiting from an NFV deployment with all functions virtualized, while for other areas, an SDN deployment with functions decomposition is more advantageous, and introduce their modeled solution for the functions placement problem.

In [24], the authors argue that both SDN and NFV are part of a bigger networking picture, that of the complete lifecycle of the network devices and therefore could take advantage of the definition of a common abstraction model, both for the forwarding model and for the network functions. Such a model will allow interoperability and homogeneity, as well as one protocol, for control, management and orchestration of the network data path and the network functions respectively. This paper proposes, defines and designs a reference Network Abstraction Model based on a building block approach, and the authors demonstrate an initial proof-of-concept implementation.

Service chaining is a key part of the NFV and SDN architectures. Service chaining is also known as network forwarding path. In NFV, a Virtual Forwarding Graph description can include zero, one or more network forwarding paths. A network forwarding path associates criteria (user profile, packet header...) to a path through VNFs to be followed by traffic flows matching the criteria (cf. Figure 24 for illustration).

![Virtual forwarding graph for chaining network services in a NFV architecture](image)
Several research works have leveraged the SDN concept to chain or redirect network traffic toward specific components. While it has been used so far to chain middleboxes in classical networking architectures, it can be as well used to chain virtualized network functions (VNF) in a NFV paradigm. One of the main applications of service chaining studied so far is its ability to redirect traffic toward components that enforce network or security policies, which is also one of the main objectives of DOCTOR. Qazi and al. put forward SIMPLE [25], an SDN-based policy enforcement layer for traffic steering. In their work, middleboxes composition and routing policies are expressed by network operators and then enforced by translating those policies into flow rules that are pushed into the network. SIMPLE uses flow correlation in order to deal with flow mangling due to dynamic middlebox actions. In the same vein, Fayazbakhsh et al. propose Flowtags [26], an architecture where middleboxes are extended to support Openflow and use tags in network packets for determining how to treat and route the corresponding traffic. Using Flowtags, network-wide policies are first expressed in a dedicated language, and then mapped to a set of flow rules and network tags to be deployed on the switches and the middleboxes in the network. Alternatively, Anwer et al. put forward Slick [27], an SDN-based architecture where the data plane can be extended with middleboxes implemented on programmable resources such as FPGA.

Stratos [28] is an orchestration layer for virtualized middleboxes in the cloud (which can be compared to VNF with proprietary interfaces) and another possible solution for dynamically composing services. Like SIMPLE and Flowtags, Stratos uses SDN-based technologies to configure the forwarding plane in a centralized way. However, Stratos is aware of its virtualized environment to address flow mangling due to dynamic middleboxes and is able to perform elastic scaling and VM migration in addition to traffic engineering. In the same field, Mehraghdam et al. have recently proposed a formal model for the chaining of VNF, as well as a methodology for finding their best placement in an operator network [29]. In [30], authors present a synthetic view of the current state of network service chaining, considering both SDN and NFV technologies, with an emphasis on how dynamic chaining can accelerate the design, implementation and deployment of novel service while reducing the costs and on the related research challenges (programming, deployment, security, etc.). In particular, questions regarding security and monitoring of the chained services are fully in line with DOCTOR’s objectives.

As we can see in the aforementioned papers, NFV in combination with SDN for the control plane comes with a lot of promises for deploying flexible, programmable and scalable virtualized environments, but key challenges related to monitoring functions for performance analysis and security should not be neglected. The Section 5 then details the functional and technical requirements for addressing those challenges in the context of the DOCTOR project.
5 Requirements and Challenges for DOCTOR

This section defines the requirements that are necessary for the project DOCTOR to successfully develop a flexible architecture for deploying and securing new networking services or protocols in virtualized environments. We mainly focus our analysis on the deployment of the NDN protocol as a use case for such a virtualized architecture.

5.1 Requirements and Challenges for Monitoring Network Traffic in Virtualized Environments

This section presents a global view of the need for monitoring virtualized networking environments in order to guarantee security and efficiency of the overall system, as well as a summary of the different possibilities for deploying monitoring tools in those virtualized networks. Finally, a list of requirements, features and challenges to be faced when deploying this kind of technology in real-world deployments is presented.

5.1.1 Definition and Scope of Monitoring

Network monitoring is a laborious and demanding task that is vital for the network infrastructure, and consists of observing the input and output events of an implementation at runtime without disturbing the normal operation of the protocol, application or service under analysis. The record of the event observation is called an event trace, which is analysed and compared to the specification or Service Level Agreement (SLA) in order to determine the conformance relation between the theoretical behaviour and the actual measurements.

The main objectives of monitoring are functional verification of the system under test, performance analysis, verification of security properties, and detection of security vulnerabilities and attacks.

In order for a monitoring platform to be effective in the early detection of any performance or security problems, several requirements must be fulfilled [31]:

- Traffic capturing performance: traffic speed and traffic volume should not affect the collection of network information.
- Extensibility: it should be easy to incorporate new services to be monitored without requiring a great effort from the administrators.
- Scalability: the monitoring system should be adaptable to the increase of traffic data and network devices. This can be done by reducing the traffic information collected through the use of efficient packet capturing mechanisms and traffic pre-processing techniques.
- Near real-time operation: in order to achieve early detection of performance or security issues, traffic capture must be aligned with real-time analysis capabilities.
- Granularity: the monitoring system should be capable of differentiating among the different protocols and services that are monitored.
- Diversity: diverse network devices, protocol stacks, services, applications, etc. should be supported by the monitoring platform.
- Low cost: the amount of computing, storage and communication resources consumed by the monitoring system should be as low as possible.
- Security: the monitoring system should not add new vulnerabilities to the network or disturb its normal operation.

Monitoring systems are usually based on probes that can be placed at any point in the network (e.g., host, router, server) so providing a great flexibility [32]. There exist two types of probes, active and passive. Passive probes do not interfere with the network and just sniff traffic to be further analysed. Active probes also analyse the network flows but allow applying automated remediation strategies that affect the system operation in terms of network and processing usage. The combination of the information provided by a set of probes offers a holistic vision of the communications and allows improved analysis of the network and system performance.
Currently, in the market, we can find different network monitoring tools with diverse purposes. Tools such as Wireshark [33] and OmniPeek [34] provide network visualization through packet analysis to allow network troubleshooting. OpenNMS [35] also offers service polling, advanced data collection, event management, alarm triggering and notifications. Other tools go one step beyond and add intrusion detection capabilities. Snort [36], BRO [37], Orchids [38], or Suricata [39] are some examples of tools that scan the network and are able to correlate events to detect, apart from network performance issues, security attacks that compromise the normal network operation.

5.1.2 Monitoring in Virtualized Environments

SDN brings better flexibility for managing the network and assuring expected confidentiality, privacy, QoS and QoE defined by Service Level Agreements (SLA).

One of the main advantages of SDN is that it simplifies network management, and facilitates the upgrade of functionality and debugging. SDN enabled centralized control and coordination makes it possible to deliver the state and policy change more efficiently, and deploy corrective measures more rapidly. Network Function Virtualization (NFV) also brings advantages since it improves scalability of applications, such as monitoring, and introduces virtualized abstraction where the complexity of hardware devices is hidden from the control plane and SDN applications. Furthermore, networks can be divided into virtual networks that share the same infrastructure but are governed by different SLA policies. SDN and NFV make possible the sharing, aggregation and management of available resources, enable dynamic reconfiguration and changes of policy, and provide granular control of network and services through the abstraction of the underlying hardware [40].

Monitoring is a solution that is required to ensure the correct operation of the whole system. Malfunctioning or even minor problems in a virtual machine could introduce vulnerabilities and instability of other virtual machines, as well as the integrity of the host machine. In DOCTOR, the monitoring function is needed to be able to precisely understand what is going on in the network, with a twofold objective. First, it is necessary for improving the security in the communications and services offered by the virtual environments. Second, from the administration and management’s point of view, it will help ensure the environment’s health and guarantee that the system functions as expected.

Existing monitoring solutions to assess security and performance can still be used in SDN and virtualized network environments. Nevertheless, existing solutions need to be adapted and correctly controlled since they were meant mostly for physical and not virtual systems and boundaries, and do not allow fine-grained analysis adapted to the needs of SDN and virtualized networks. The lack of visibility and controls on internal virtual networks, and the heterogeneity of devices used make many performance assessment applications ineffective. On one hand, the impact of virtualization on these technologies needs to be assessed. For instance, QoS monitoring applications need to be able to monitor virtual connections. On the other hand, these technologies need to cope with ever-changing contexts and trade-offs between the monitoring costs and the benefits involved. Here, virtualization as well as SDN facilitates changes, making it necessary for monitoring applications to keep up with this dynamicty.

Solutions such as Ceilometer [41], a monitoring solution for OpenStack, provide efficient collection of metering data in terms of CPU and network costs. However, it is focused on creating a unique contact point for billing systems to acquire all of the measurements they need, and it is not oriented to perform any action to try to improve the metrics that it monitors. Furthermore, security issues are not considered.

StackTach [42] is another example oriented to billing issues that monitors performance and audits the OpenStack’s Nova component. Similarly, but not specifically oriented to billing collectd [43] gathers system performance statistics and provides mechanisms to store the collected values.
A recent project from OPNFV\textsuperscript{1}, also named Doctor [44], is focused on the creation of a fault management and maintenance framework for high availability of network services on top of virtualized infrastructures. The main difference with the DOCTOR project resides in that OPNFV-Doctor only focuses on performance issues and does not consider security aspects.

In terms of security, OpenStack provides a security guide [45] providing best practices determined by cloud operators when deploying their OpenStack solutions. Some tools go deeper in order to guarantee certain security aspects in OpenStack, for instance: Bandit [46] provides a framework for performing security analysis of Python source code; Consul [47] is a monitoring tool oriented to service discovery that also performs health checking to prevent routing requests to unhealthy hosts.

The monitoring solution proposed in DOCTOR, Montimage Monitoring Tool (MMT), offers a global monitoring and security approach by providing advanced monitoring to audit QoS and performance, and use this data to trigger security alarms and countermeasures, so enhancing the user experience by ensuring that the SLA terms are always fulfilled.

Figure 25 shows a conceptual deployment of the Montimage Monitoring Tool (MMT) in an SDN environment in which MMT probes are co-located in the virtual machines with the network functions (e.g., ICN/CCN nodes). This figure represents one of the possible options for the monitoring deployment. In the following section diverse alternatives are described as well as their benefits and drawbacks.

---

\textsuperscript{1} https://www.opnfv.org/
5.1.3 Where to Implement Monitoring Functions

NFV introduces virtualized networks and functions that need to be monitored. To be able to assure end-to-end QoS and security, a monitoring architecture needs to be defined and deployed in order to measure and analyse the network flows at different observation points that could include any component of the system, such as physical and virtual machines. Setting up several observation points will help to better diagnose the problems detected. With SDN it is possible to create network monitoring applications that collect information and make decisions based on a network-wide holistic view. This enables centralized event correlation on the network controller, and allows new ways of mitigating network faults.

The monitoring probes can be deployed in different points of the system. Let’s consider a single hardware entity that is controlled by a hypervisor that manages the virtual machines. A first approach consists of installing the monitoring solution (MMT) in the host system (hypervisor) that operates and administers the virtual machines (see Figure 26), in this way providing a global view of the whole system. This approach requires less processing power and memory to perform the monitoring operations, since the protection enforcement is located in a central point. In this way, network connections between the host and the virtual machines can be easily tracked allowing early detection of any security and performance issue. The main problem of this approach resides in the minimum visibility that the host machine has inside the virtual machines, not being able to access to key parameters such as the internal state, the intercommunication between virtual machines, or the memory content.

![Figure 26: Network-based protection](image)

Monitoring probes can also be located in a single privileged virtual machine that is responsible for inspection and monitoring of the rest (see Figure 11). This approach is called Virtual Machine Introspection (VMI) and offers good performance since the monitoring function is co-located on the same machine as the host it is monitoring and leverages a virtual machine monitor to isolate it from the monitored host [48]. In this way, the activity of the host is analyzed by directly observing hardware state and inferring software state based on a priori knowledge of its structure. VMI allows the monitoring function to maintain high levels of visibility, evasion resistance (even if host is compromised), and attack resistance (isolation), and even enables the manipulation of the state of virtual machines. Unfortunately, VMI based monitoring software is highly dependent on the particular deployment and requires privileged access that cloud providers need to authorize.

![Figure 27: Virtual machine introspection](image)
The approach that offers the best security performance is the deployment of the monitoring tools in every virtual machine. In this way robust protection can be achieved since the security software has a complete view of the internal state of every virtual machine, as well as the interactions with the host or any other virtual machine. Figure 28 shows how this approach can be deployed.

Figure 28: Host-based protection

This third solution offers the best performance in terms of security. Here, the processing power and memory required are distributed among the virtual machines. Furthermore, its deployment is simpler than other approaches since it can be included in the software image of the virtual machine, so it is automatically initiated when instantiating each virtual machine with no further configuration needed.

Despite of the individual probes installed on each virtual machine, there is the need of a global monitoring coordinator that supervises the monitoring tasks of each probe installed on each virtual machine. For this, each probe must be able to directly interact with any other probe, as well as with the monitoring coordinator. Local decisions can be taken by the individual monitoring probes installed on each virtual machine, and the monitoring coordinator can perform coordination, orchestration and complex event detection.

5.1.4 SDN-based Monitoring Architecture

Considering the different monitoring deployments presented in the previous section, herein, a whole architecture integrating monitoring probes and coordinator is presented.

Figure 29 represents a possible deployment scenario for MMT in an SDN environment. As depicted, MMT probes capture performance and security meta-data from each virtual machine, and are able to perform countermeasures to mitigate attacks and security risks. MMT probes have the capacity of P2P communication, so they can share relevant information with the aim of increasing the efficiency of the security mechanisms and, thus, ensure the correct operation of the whole system. To perform coordination and orchestration of the whole monitoring system, a central MMT Operator will receive information from the distributed MMT probes. The MMT Operator is also in charge of correlating events to create reports to inform network managers of the system activities, attacks avoided and countermeasures taken. Furthermore, it will be able to globally analyse the information provided by individual MMT probes with the ultimate objective of detecting complex situations that may compromise the system.

The architecture detailed in Figure 29 shows the deployment of MMT (MMT probes and MMT Operator) over a set of physical hardware platforms that can be part of one or several cloud service providers. The MMT Operator will be in charge of coordinating the diverse probes deployed in each virtual machine and provide a global view.
5.1.5 Coordinating Security and Traffic Management

Network security must be considered parallel to the traffic management and as an integral part of the network management. In SDN, due to its dynamic nature, this becomes even more critical since a security lapse of the centralized controller will adversely affect setting the traffic flow rules on other data path elements managed by the same controller. Similarly, network security policies and procedures must be updated in the case there are changes in the architecture or topology. The combination of proactive and reactive techniques synchronized with traffic management leads to a better security framework. Proactive operations help minimizing security lapses and service degradation; thus favoring a controllable usage of resources. Reactive mechanisms have to be adequately located to avoid an impact on the performance even when providing a fine-grained control to access the resources. Synchronization in SDN is possible since the controller has a global view of all the network elements, flows and their security requirements and all the security policies. It is necessary to assure, for instance that: the traffic from data-path elements does not stop due to controller failure because of attacks through other data-path elements under the jurisdiction of the controller; and, the security (e.g., access control and firewalls) is configured and active as soon as the flows are directed to new routes (e.g., due to changes in topology).

5.1.6 Requirements and Features of the SDN-based Monitoring System

The monitoring architecture presented in the previous section 5.1.4 requires addressing several constraints and challenges related to its deployment in SDN and NDN based solutions. In this section we list the main requirements and provide more details on the monitoring interfaces that are needed, the metrics that can be measured, and the countermeasures that can be taken to improve the performance and security of the system. Finally we list the main challenges that need to be addressed to obtain an effective and useful monitoring system in the SDN/NDN context.

5.1.6.1 Main Monitoring Requirements

The main requirements and features of the monitoring system when using an SDN-based architecture are:

- Monitoring probes will be included in the operating system images that will be launched in each virtual machine.
- Monitoring probes need independent and secure communication capabilities so they can communicate among themselves in a P2P fashion and with the monitoring coordinator.
- Monitoring probes require complete access to those parts of the system (e.g., virtual machine, network interfaces) where monitoring needs to be performed.
- Monitoring probes will eventually be able to trigger special actions in the virtual machine (e.g., modify the configuration of a firewall) if certain security issues are detected.
• Monitoring probes will not interfere with the normal operation of the virtual machines.
• The monitoring coordinator will be installed in a separate machine to those where the monitoring probes are operating.

5.1.6.2 Monitoring Interfaces

In order to ensure the correct operation of the monitoring system, various interfaces are needed:
• Monitoring probes need to be able to exchange information in a P2P fashion, with the aim of early detection of security and performance issues that may affect the rest of virtual machines.
• The interface between the monitoring coordinator and the monitoring probes will allow the coordinator to analyse information gathered from various probes, and also to orchestrate coordinated actions to avoid complex security and performance issues and to create reports with information regarding the performed analysis and the countermeasures taken.

Through these interfaces, a series of commands may be executed, both by the monitoring probes and by the monitoring coordinator, to obtain the required information, and to execute the countermeasures needed to guarantee the security and satisfactory performance in the NDN nodes, or in the whole system:
• Inspection commands: are used to directly examine the network and virtual machine state (e.g., link usage patterns, delays, memory and register contents).
• Monitor commands: allow to detect when certain events occur and request notifications through an event delivery mechanism (e.g., use of uncommon network ports, abnormal communication patterns, and suspicious memory accesses).
• Administrative commands: are used to trigger special actions in the virtual machines when certain security issues are detected (e.g., suspend the operation of a virtual machine, disable the network access to avoid the propagation of a security issue, raise an alarm to notify the administrator) and also commands to manage the monitoring probes.

5.1.6.3 Metrics to Be Measured

The metrics that will be measured by the monitoring deployment include, but are not limited to:
• Link usage: the number of correctly received IP-layer bits sent from any source and passing through a determined link during a fixed period of time.
• Packet delay: the time elapsed between the transmission and the reception of a communication packet.
• Jitter: the delay variability in a series of continuous packets. High jitter values may represent attacks or unauthorised access to the network.
• Flow patterns: certain changes in flow patterns may represent network malfunctions caused by attacks or unauthorised access.
• Port surveillance: port activity that could represent unauthorised access attempts.
• Access attempts to secure resources (e.g., memory, network interfaces, storage).
• Access attempts to protected information (e.g., credentials).
• Credentials edition: any unauthorised attempt to edit the system credentials must be avoided to not compromise the security of the system.
• SLA violations: any unauthorised attempt to edit the system credentials must be analysed in order to determine possible malicious behaviour.
• NDN specific metrics.

5.1.6.4 Countermeasures to Be Taken

With the aim of mitigating security issues and optimizing the performance of the NDN nodes, the monitoring system will trigger a series of countermeasures that include, but are not limited to:
• Ban the access to external connections that show a suspicious behaviour.
• Refuse incoming connections from hosts that have previously attempted to degrade the security, stability, performance, or normal operation of the system.
• Kill processes in the NDN nodes that represent a risk for their normal operation.
• Communicate, to the monitoring coordinator, global statistics and any suspicious behaviour that may represent a security or performance issue for further analysis.
• Avoid (ban) any non-authorised access to any component of the system (e.g., memory, network interfaces, storage).
• Route malicious traffic to a FIB black hole or sandbox for further analysis.

5.1.6.5 Challenges to Be Addressed

The main challenges that need to be addressed are:

• Interaction of monitoring functions with the SDN and NDN components.
• Achieve integrity protection during runtime without interfering with the normal operation of the system.
• Find the best compromise between the OPEX, the CAPEX, the performance, scope and granularity of the monitoring function.
• Introduce the distribution of the monitoring tasks and the use of virtualized resources to improve the scalability of the monitoring function.
• Assure that the monitoring system is open enough to deal with the required flexibility of NDN environments, without losing any of its functionality and required performance.
• Coordinating and orchestrating the distributed probes in order of create a secure and efficient ecosystem.
• Determine the NDN specific metrics. To identify NDN flows, for instance, we could use the content ID but this might not be enough when multiple-contents, multiple-consumers and caching make data traceability practically impossible.
• In the case of encrypted traffic, part of the monitoring can be done using the unencrypted headers and statistics that do not require any decryption. For deeper analysis (DPI), monitoring of encrypted traffic needs to be decrypted or to be analysed at the end-points.

5.2 Requirements and Challenges for Risk Assessment in Virtualized Environments

Another important point that must be addressed in order to push forward the deployment of virtualized architectures is the risk assessment. In this section, we will define the risk assessment and see how its architectures can be adapted to take into account the requirements and challenges of virtualized environments.

5.2.1 Risk Assessment

Risk Assessment is an important part of cyber security whose goal is to determine the level of security of an infrastructure or information system, facing recognized threats. It is used principally by cyber security operators working in Information Security Operations Center (SOC) and can be quantitative or qualitative, according to the needs and the assessment functions used.

Qualitative risk assessment tries to know what events or attacks threatened an information system. It can be based on a Risk Analysis (such as MEHARI (Method for Harmonized Analysis of Risk), or ISO/IEC 27005).

Quantitative risk assessment requires the estimation of two parameters:

• The probability of occurrence of the threat: what is the chance that such threat happens in the considered information system?
• The impact that may have this threat: if this threat happens, how much will it cost, or what impact will it have on my information system?

Risk Assessment can be split up into Static Risk Assessment and Dynamic Risk Assessment, according to the time period in which the risk assessment is done.
Static Risk Assessment is based on the knowledge of an information system in a static way: we know its configuration at a fixed time, and want to know the risk that faces this system. The system to be evaluated can either be a real deployed system, or a system being designed. For a deployed information system, security operators will periodically want to ensure that their information system is under controlled risk. For a system being designed, security architects want to ensure that the system they are building will be secure against the known threats that will attack it. Static Risk Assessment is often concluded with the estimation of the acceptable risk, the risk that will not be corrected, due to too high costs, very low probability of occurrence, or low impact.

Dynamic Risk Assessment is based on the same knowledge of the information system, but also needs dynamic information such as alerts issued by Intrusion Detection Systems (IDS) or Security Information and Event Management systems (SIEM), logs... Such dynamic information allows the risk assessment method to take into account the attacks that may be currently happening. The probability of occurrence of the threat can thus be updated according to the dynamic events attesting (or not) that a threat is currently happening.

5.2.2 Risk Assessment Models

Risk Assessment is often based on a risk assessment model, describing all the attacks that can happen in an information system. Attack trees are a well-known multi-step attack model which is very important since it is one of the first graphical models to have been proposed for security assessment. The notion of attack tree was introduced by Bruce Schneier in [49]. In his article, the attack trees are defined informally as a tree with AND/OR nodes which describe exactly the possible or required steps to do an attack and arcs modeling dependencies between these steps. The goal of the attack is located in the root of the tree and the basic actions used to achieve this goal are leaf nodes.

The main limitation of attack trees is that they only describe one main attack. To respond this limitation, attack graphs have been created. An attack graph is a model that regroups all the steps that an attacker may follow in an information system during an attack from both outside or inside the local network. This formalism has been widely used, thus many heterogeneous models are now behind the name attack graph. Generally, vertices (also called nodes in the literature) represent opportunities in an information system or actions that can be done by an attacker, and edges (also called arcs in the literature) represent the dependency relations between the opportunities and/or actions. An attack graph can be built using information about the potential exploits that can be carried out on a network or using existing vulnerabilities databases. A summary of the state of the art on the early papers about attack graphs (from 2002 to 2005) has been done by Lippmann and Ingols in [50]; a more recent by Kordy et al. in [51].

MulVAL, The Multi-host, Multi-stage Vulnerability Analysis Language Tool [52] is an open source attack graph engine. It was built in Kansas University and is based on the logical programming language Datalog. MulVAL is constituted of two components:

- A scanner: It has to be launched on each host and can use a vulnerability database and extract the local configuration to deduce the vulnerabilities on this host.
- A centralized analyzer: It is launched as soon as new data are collected from a scanner. It analyzes the data that was gathered and generate the corresponding attack graph.

MulVAL can use six types of inputs to generate the attack graph: the vulnerabilities existing on each equipment, the software and configuration, the network configuration, the user accounts on the hosts, the authorized interactions between the hosts, and the security policy. The attack graph generated with MulVAL is a logical attack graph (each node of the attack graph represent a logical fact). MulVAL is available as an open source software and is licensed under GNU GPL License v3.
5.2.3 Risk Assessment Models Inputs

In order to build the risk assessment models described previously, and do a proper risk assessment, several inputs may be needed. Some inputs are required, other are optional, to get better results.

- **Vulnerability (required):** The first input that is necessary for most risk assessment models is an inventory of the vulnerabilities of the assets of the information system. This information is generally collected with a vulnerability scanner (for example Nessus, Openvas…), but can be enriched thanks to the knowledge of operators or security experts. The vulnerability scanner should be preferably launched with local access on each machine, rather than as a remote scan, because the results of a local scan are more accurate, but can be enough.

- **Network topology (required):** Even, if such information can be deducted from a scan, it is much more accurate, when the network topology is given as input of the risk assessment models. The network topology describe the hosts of the information system, with their hostname, interfaces, IP addresses, VLANs belongings,…

- **Flow matrix (required):** This input is the most difficult to get in classical infrastructure, as it is generally not directly collectable. The flow matrix describes the flow that are authorized (and possible) in an Information System. It is a high-level view of the firewall rules that are deployed in the information system. The flow matrix describes which accesses are possible between hosts or networks.

- **Network services description (optional):** This input can also be deducted from a vulnerability scanner report, but it is always more accurate to specify it, when this information is available. The network services are those which are deployed on hosts of an information system, and that listens to connections from the outside. These services are particularly exposed, as they can be targeted by attackers from the outside of the hosts, and could allow an attacker to take the control of the service, if it is vulnerable. The description of such services can be a list of the open ports and services attached on each host, or a more complete list, with the software and versions installed for each service.

- **Business inputs (optional):** The business inputs are not necessary, but allow to prioritize strongly the risk assessment of an information system, as they describe the assets that are a priority for the business running on the information system. These inputs are necessary to determine the impact of the threats on the information system. The business information can be attached to hosts or more accurately to services.

- **Alerts and detections (required for DRA):** For Dynamic Risk Assessment is also required to have alerts issued by the sensors corresponding to some conditions of attacks. Such alerts may be given with a confidence index (for example for SIEM alerts) which improves the analysis.

5.2.4 Risk Assessment Architectures

The usual architecture of risk assessment is centralized: a centralized scanner collects the vulnerabilities that are present on hosts of the information system, which are added into a central database that also contains the topological data. All this information is then used by a centralized risk assessment engine. For Dynamic Risk Assessment, the assessment engine also connects to a log sink (for example a SIEM, or a central log database) that centralized the logs of all equipment. Such architecture can be seen on the following Figure 30.
The default of such centralized infrastructure is the difficulty to have an updated topology database, having all the information about new hosts, services, access control list... Thus, another agent-based hybrid architecture is also possible for risk assessment. This architecture is more adapted to fit with virtualized environment, as it is much easier in such environments to automatically deploy an agent on each host, to collect the topology (and eventually the vulnerabilities). Such architecture can be seen on the following Figure 31.

The last architecture that is conceivable for risk assessment is a truly distributed architecture, where each node computes local risk assessment and shares it with the other nodes. It may be an interesting architecture for virtualized environment, if the control part is also distributed.
5.2.5 Challenges and Advantages of Risk Assessment in Virtualized Environments

There are several challenges of doing Risk Assessment in virtualized environments:

- The first challenge is due to the highly-changing topology of such an infrastructure. Indeed, virtualized functions can be deployed dynamically and thus change significantly the network topology (routing changes, new firewalls…). The risk assessment depending on such a topology has to take into account these changes.

- Another challenge of such infrastructure is the new vulnerabilities that may exist. Virtualization supposes that proper isolation is done between layers, but if a vulnerability exist in the software isolation, new vulnerabilities may appear.

There are also challenges for Risk Assessment related to NDN the use-case, especially for the Dynamic analysis:

- Data traceability along its lifecycle: in transit, or at rest. Indeed, in such protocol, the same data can be stored in several places (content routers, content store…), it can also exist in several versions and we must know precisely where is which data, by being able to list data stored in cache.

However, virtualizing the infrastructure and reposing on SDN brings some new advantages comparing to classical infrastructures:

- Easy access to the network topology in real-time. In SDN, the controller part has access to a whole view of the network topology and the actual configuration that is deployed. This component will thus be able to give to the risk assessment engine a real view of the configuration of the system.

- Easier to deploy agents. A problem of the hybrid architecture is the difficulty to deploy agents on all hosts, for classical infrastructures. For virtualized ones, it will be possible much easier to deploy such agents, for example by implementing a common base template for all virtual components.

- NDN deployed in such infrastructure will allow to sometimes anticipating the dissemination of sensitive or malicious data by observing Interests in the network. It can also be used to detect the propagation of DoS attacks, by analyzing the spread of increasing requests through a set of NDN routers.

5.3 Requirements and Challenges of Remediations Deployment in Virtualized Environments

In order to decrease the risk level and improve the security of an architecture, remediations may be taken. In this section we will define the remediations and present the requirements and benefits of the ones that take into account the specificity of virtualized environments.

5.3.1 Remediations

Remediations to an attack can be regrouped in three types: corrective, active and passive.

The first one is the correction of the exploitable vulnerability. This is generally implemented by patch management software. The technology is quite mature (several tools exist, vendors regularly propose patches for their software) but it suffers from limitations detailed by Cavusoglu et al. in [53]. The most important is that patch deployment still requires human intervention: each patch must be tested on all platforms to prevent conflicts or regressions before being applied.

The active remediations regroup those that prevent the exploitation of a vulnerability that still exists after the deployment. This is for example the case of simple filtering by a firewall or an Intrusion Prevention System (IPS). An IPS blocks flows that have been flagged as abnormal thanks to a signature or due to its statistical behavior [54]. More generally, an Intrusion Response System (IRS) is a system that provides other types of responses to a detection. This is a currently active research topic and many papers treat this subject, as summarized by Shameli-Sendi et al. in [55].
Finally, the passive remediations regroup the detection of the exploitation of a vulnerability and its report. This is a last resort but is widely used, as it can help security operators to know what happened in their information system. A system that only provides passive responses (alerts, reports, logs...) is generally called Intrusion Detection System (IDS) [56].

5.3.2 Types of Remediation

Several types of remediation exist and some could be specific to virtualized network architectures:

- **Patch of a vulnerability:** This is one of the first remediations that come in mind to correct a vulnerability. A patch is a piece of software that corrects a vulnerability in a bigger software. This is a long-term remediation as it corrects the vulnerability that is no longer exploitable. However, sometimes no patch exists for a vulnerability, and patch deployment needs often manual intervention and can interrupt service, if a restart is required.

- **Firewall reconfiguration/deployment:** When we want to prevent a flow in a network, it is possible to reconfigure/deploy a firewall, in order to add a firewall rule blocking such a flow. This can sometimes prevent the exploitation of a vulnerability. However, before deploying such remediations, we must ensure that they do not prevent legitimate flows.

- **Intrusion Prevention System (IPS) reconfiguration/deployment:** An Intrusion Prevention System, allows to block flows when they match a specific IPS rule. By reconfiguring/deploying an IPS, it is possible to activate new rules that prevent an attacker to exploit a vulnerability.

- **Routing reconfiguration:** It is also possible to use routing to remediate some vulnerabilities. Indeed, by reconfiguring routing, it is possible to redirect some suspicious flows to a black hole, or to another equipment that can make a deeper analysis. This is for example particularly used for DDOS mitigation.

- **DPI probe reconfiguration/deployment:** A last network remediation is to reconfigure or deploy Deep Packet Inspection probes (for example a MMT probe). Such probes rebuild packets routing through them, until the application layer, in order to apply specific application-layer rule. For example, this can be used to rebuild file on transit on HTTP, and block specific files.

5.3.3 Simulation of the Network Topology

The ability to simulate the network topology may be very useful to compute remediations. Indeed, as detailed in [57], to estimate the impact that may have a remediation on a whole information system, it is necessary to apply this remediation on a simulated network topology, and see if it has a negative impact on genuine services. This simulation is generally quite difficult to implement in practice, as we need a complete description of the network (routing, firewalling, NAT...) of what is currently deployed on the actual system.

5.3.4 Challenges and Advantages of the Deployment of Remediations in Virtualized Environments

There are several challenges that may happen to deploy remediations security functions in virtualized environments:

- **Difficulty to orchestrate the network plane:** One main challenge of the application of remediations through the reconfiguration or deployment of virtualized security functions is the orchestration of the network plane that will be changed by remediations. Indeed, such an environment is very complex, and if it is the remediation computation engine that needs to orchestrate the entire network plane, it will have to deal with the complexity of the remediations in addition to the network complexity. If it is an external orchestrator that manages the network plane, the remediations should have the ability to change the configuration, while keeping using such an external orchestrator.

- **Keep a remediation along time:** One other problem that may arise is due to the fact that virtualized environments evolve very fastly. In such systems, we must ensure that the deployment of a remediation will not be replaced by a new configuration adapted to a topological change, which will make the system vulnerable again.
Know the remediation features provided by virtualized functions. It will also be a challenge to describe the remediation features that will be provided by each virtualized functions and virtualized infrastructure management technologies.

However, virtualizing the infrastructure and reposing on SDN bring some new advantages comparing to classical infrastructures to deploy such remediations:

- Ability to collect up-to-date configuration: These virtualized technologies will give us the ability to know in real time the configuration of the network, in order to be able to simulate the network topology, and estimate a remediation impact.
- Deployment of new types of remediation: In a virtualized infrastructure, where virtualized network functions can be deployed on demand, new types of remediation will be available. For example, in classical infrastructure, deploying a new firewall or DPI probe is a long-term remediation, whereas in virtualized infrastructures, it can be done in few seconds.
- Benefiting from centralized information: SDN will also allow us to get a central point which will know the topological information, as well as some business related information (applications that require some flows, priorities...).
- Facilitate the simulation of network topology: By using the facility to collect topological information, and the ability to quickly deploy new systems, it will be much easier to simulate a very accurate representation of the network topology, in order to see the effects of remediations.
- NDN-based remediations: For the NDN use case, it will be possible to deploy specific remediations, preventing some attacks. For example, blacklisting compromised NDN nodes (using their public key identifier), or asking for the deletion of data identified by its Content Name, in the Content Store, will prevent the propagation of malware or sensitive data propagation.
6 Assessment of the Requirements and Possible Candidates

The Section 3 presents a panorama of server virtualization techniques in data centers, and the Section 4 provides their technical advances applied to the networking data plane, with respect to the concept of Network Functions Virtualization (NFV) leveraging Software Defined Networking (SDN) for getting the networking control plane more agile and programmable. The Section 5 describes the different requirements and challenges for the project DOCTOR to design a NFV-based architecture for deploying new networking services, such as the NDN delivery service, in virtualized environments, while delegating the network control to a SDN controller for routing configuration and network monitoring, making it possible to secure the overall virtualized architecture (cf. Figure 32).

In this section, we map the DOCTOR overall architecture into a layered architecture based on our analysis in the previous sections (cf. Figure 33). If this architecture needs to be refined in task 1.2, it enables to have a good knowledge of the main functions to provide and the kind of solutions we envision so as to offer the SDN-enhanced NFV framework. One can note that the Doctor project requires and gathers many technological frameworks that may be at an early stage of development for some of them (e.g. NFV implementations). In this context, it is acknowledged by the Doctor consortium to (1) build an incremental infrastructure rather than implementing an holistic overall architecture at first and (2) follow any updates of components by a careful and regular technological watch.

![Figure 32: Overall virtualized architecture for the project DOCTOR](image-url)
6.1 Infrastructure Layer

The Infrastructure layer consists of standard commodity hardware based on x86 servers on which virtualization is applied to network services of the Application layer. The equipment can be HP, IBM or DELL machines. For instance, the UTT part of the tested infrastructure will be deployed over some Dell R720 and R720xd servers interconnected with a gigabyte L2 backbone (and 10Gbps for R720 servers).

6.2 Virtualization Layer

This layer includes the virtualization system we will use for the DOCTOR project. Based on the requirements we have listed before in the previous sections and those inherent to defined use-case (deployment of NDN), we believe that an OS-level virtualization (type 0) is the most suitable. Amongst the candidates, Docker seems to be the most promising one. The Docker Engine container comprises just the application and its dependencies. It runs as an isolated process on the host operating system, sharing the kernel with other containers. It also allows direct access to the hardware resources than other techniques such as the network interface, for instance, a strong requirement for our NDN use-case requiring high throughput for processing packets. However, the choice of the virtualization technology will be made compatible with the one of the control and management part of the Doctor overall infrastructure in order to provide a homogeneous and fully interoperable framework. For instance, the actual compatibility between Docker and OpenStack must be assessed to confirm or decline the choice of such a technology. Also, backup solutions such as KVM for instance will be considered in case of technological barriers so that the project unfolding is not impacted.

6.3 Application Layer

The Application layer includes the network functions and services we target in the project. The first one is the NDN protocol stack we aim to adapt to be deployable into such a virtualized environment. For this, the NDN components will be deeply investigated to determine if they can be split into smaller entities (e.g.; the CS, FIB and PIT), and then be deployed and chained to offer the overall service. It will be one major activity in task 1.2.

In this layer, we will also have the monitoring components, which will analyse the traffic as well as all other information adapt required to provide our secure and reliable solution.
Together with the monitoring, the security components will be deployed into this layer so as to detect abnormal behaviour or attacks (e.g., DDOS), and activate the counter-measures. This can be done in a proactive manner or reactive one. We also plan to deploy a NDN firewall, in relationship with other security components.

### 6.4 Control Layer

The Control layer in the DOCTOR architecture consists of configuring, controlling, managing the virtualised environment as well as the VNFs. For controlling the virtualised environment, a northbound Application Programming Interface (API) is defined and might be implemented using the OpenStack Neutron component, enabling applications and orchestration systems to program the network and request services from it.

For controlling the VNFs themselves, a southbound interface is defined and might be implemented using OpenFlow and OpenDaylight. The OpenDaylight project, hosted by the Linux Foundation, proposes an implementation of a SDN controller capable of communicating with OpenStack and using OpenFlow as the southbound API. Southbound APIs facilitate efficient control over the network and enable the SDN Controller to dynamically interact with the forwarding plane (implemented as virtualized network functions in the NFV context) to make adjustments to the network, so it can better adapt to real-time demands and needs (e.g. adding a black hole route in a NDN router for redirecting malicious traffic).

The following lists some possibilities the control layer can offer, which will be reassessed in light of further work in the next steps of the project:

- The SDN principles can be applied to dynamically configure the Forwarding Information Base (FIB) tables of the NDN routers. Indeed, since we imagine to use jointly NDN and IP, NDN being used for some specific services, web sites or regions. In this case, the programmability of the NDN routers via a controller, using the SDN concept, can be very useful to improve the NDN configuration. Between the controller and the NDN routers, we can rely on the OpenFlow protocol, but extended to manage NDN names instead of IP addresses.

- The SDN architecture can also be defined to configure some virtualized components running in our DOCTOR environment. Focusing on the NDN use case and for ensuring security, we plan to deploy a NDN firewall. We can then imagine being able to dynamically configure the firewall rules, based on specific events or network conditions. This information can, for instance, be provided by the monitoring tools or be the result of a security analysis performed by the security module. SDN is in this case, seen as an interface between the NDN firewall running in the virtualized environment and a controller outside of the node.

- The DOCTOR project aiming at designing an overall secure networking architecture (with pro-active actions but also reactive actions), SDN can be used to configure the security actions to take in order to secure the infrastructure. The security manager or controller, which has a knowledge of possible (or active) attacks and of the current network configuration, can decide to change some policies, some rules, regarding the traffic passing through the nodes, the routing/forwarding between the nodes, etc. Those actions can be configured via a SDN-like approach.

- Monitoring the network is a critical task for network operators. It is the basis for the security task but it is also valuable for the network knowledge (network load, type of traffic, peak hours). We can imagine to use SDN to dynamically program the monitoring modules (i.e. the MMT probes described in Section 5.1) and inform them to monitor in more details specific information, or to aggregate specific information, usable by a monitoring platform (the MMT operator in Figure 29), etc. This openness of the monitoring module can bring dynamicity and personalization of the monitoring stuff for the network operators.
7 Conclusion

The present deliverable D1.1 is the first technical deliverable of the DOCTOR project. This document first exposes an overview of current virtualization techniques and their application to the Network Functions Virtualization (NFV) paradigm. Then, the document describes the requirements and challenges for the DOCTOR project whose objective is to propose a flexible virtualized architecture that can be used to deploy new network services within well-defined network monitoring policies for ensuring security of the overall architecture. Based on those requirements and the current advances in open and standardized interfaces between physical hardware, virtualization and network, as pushed by the SDN concept, we also present in this document our first analysis for the DOCTOR virtualized architecture that includes the physical layer, the virtualization layer, the application layer and the control layer, along with the possible candidate technologies for its implementation.

It is worth of noting that this document establishes an initial set of requirements as well as candidate technologies that will be rigorously updated during the project lifetime in different WPs. The architectural process needs to be an iterative and incremental approach. Our first candidate architecture will be a high-level design that we can test against our NDN use case, related requirements, known constraints or issues. As we refine our candidate architecture, we could learn more details about the design and could be able to further expand our design with new resulting requirements and improve our approach to address emerging issues.
8 References


[8] A. Elsayed and N. Abdelbaki, "Performance evaluation and comparison of the top market virtualization hypervisors", in Proceedings of ICCES 2013 (Cairo), 978-1-4799-0080-0/13/$31.00 ©2013 IEEE.


Deliverable D1.1: Virtualization Techniques: Analysis and Selection


[34] http://www.wildpackets.com/products/


[38] http://www.lsv.ens-cachan.fr/Software/orchids/


[41] http://docs.openstack.org/developer/ceilometer/


