D4.1
Preliminary Architecture of the Multi-Cloud Network Virtualization Infrastructure

Project number:	643964
Project acronym:	**SUPERCLOUD**
Project title:	User-centric management of security and dependability in clouds of clouds
Project Start Date:	1st February, 2015
Duration:	36 months
Programme:	H2020-ICT-2014-1
Deliverable Type:	Report
Reference Number:	ICT-643964-D4.1/ 1.0
Work Package:	WP 4
Due Date:	Oct 2015 - M09
Actual Submission Date:	2nd November, 2015
Responsible Organisation:	FFCUL
Editor:	Fernando M. V. Ramos, Nuno Neves
Dissemination Level:	PU
Revision:	1.0
Abstract:	In this document we describe the preliminary architecture of the SUPERCLOUD multi-cloud network virtualization platform. We define its requirements, review the state-of-the-art, and present a first design of the proposed architecture.
Keywords:	network virtualization, multi-cloud, software-defined networking, self-management, security

This project has received funding from the European Unions Horizon 2020 research and innovation programme under grant agreement No 643964.

This work was supported (in part) by the Swiss State Secretariat for Education, Research and Innovation (SERI) under contract number 15.0091.

Editor

Fernando M. V. Ramos, Nuno Neves (FFCUL)

Contributors (ordered according to beneficiary numbers)

Marc Lacoste, Nizar Kheir (ORANGE)
Max Alaluna (FFCUL)
Fabien Charmet, Khalifa Toumi (IMT)

Disclaimer

Executive Summary

In this document we present the preliminary architecture of the SUPERCLOUD multi-cloud network virtualization platform. We start by defining the requirements of the architecture to then review the state-of-the-art. Namely, we survey Software-Defined Networking (SDN), the paradigm used in the proposed solution, and discuss recent SDN-based network virtualization platforms that offer insight into shaping the proposed design. As security is a first-class citizen in our architecture, we present a survey on security incidents involving public cloud services that motivates the need for self-management of network security. The document closes with the preliminary design of the SUPERCLOUD network virtualization platform with a focus on how the proposed architecture fulfills the requirements set as goal.

Contents

List of Figures

List of Tables

Chapter 1 Introduction

The management of cloud computing infrastructures was made possible by the advent of server virtualization. By exposing a software abstraction (the Virtual Machine, VM) to cloud users instead of the physical machine itself, virtualization has given the degree of flexibility necessary for providers to achieve their operational goals while satisfying customer needs.
The requirements of today's cloud users – for example, the ability to migrate unchanged workloads to the cloud – cannot be met with server virtualization alone. The root of the problem is the fact that, although compute and storage virtualization is commonplace, network virtualization is not. Complete network virtualization, as required to offer Network-as-a-Service, entails fully decoupling the logical service from its physical realization [20]. Traditional network virtualization primitives, such as VLANs, are too coarse-grained and lack the scalability to provide this form of virtualization [76].
Software-Defined Networking [47] (SDN) is an emerging paradigm that promises to overcome the limitations of current network infrastructures. By separating the network control logic (the control plane) from the underlying routers and switches that forward the traffic (the data plane), network control is logically centralized, simplifying policy specification and enforcement. SDN is thus seen as a natural platform for network virtualization [25], and recent efforts have indeed started using this technology to fully virtualize networks [44, 9].

1.1 Challenges

Recent state-of-the-art network virtualization platforms [44, 9, 25] are an important step towards full virtualization, but they have limitations when considering the SUPERCLOUD deployment scenarios.
The first problem is that these solutions target a single datacenter/cloud provider. This dependency of the user on the cloud provider is a fundamental limitation in terms of scalability, interoperability, security, and dependability. First, a user may want to run its workloads in one cloud (or on its own on-premise cluster), but automatically overflow to its cloud-hosted cluster(s) if it runs out of on-premise capacity. Second, in order to avoid vendor lock-in, the user may want its workloads to run across multiple cloud providers all the time. Third, as some workloads may be privacy-sensitive or have other security requirements, they should be automatically diverted to run in the user's secure, on-premise cluster. Finally, the user may want to be immune to any single datacenter or cloud availability zone outage, and in this case he or she would like to spread the services across multiple cloud providers. Indeed, a large number of incidents involving accidental and malicious faults in cloud infrastructures [51] offer evidence that relying on a single provider can lead to the creation of internet-scale single points of failures for cloud-based services.
The second problem is that existing SDN-based network virtualization solutions do not consider security and dependability in their design. The main benefits that make SDN a suitable architecture for network virtualization – network programmability and control logic centralization – open the doors for security threats and dependability concerns that did not exist before [46]. Indeed, traditional networks have "natural protections" against network attacks, including the closed (proprietary) nature of network devices, the heterogeneity of the software, and the distributed nature of the control plane. These represent defenses against common vulnerabilities. This level of protection is comparatively smaller in SDNs, increasing the surface for malicious and non-malicious attacks.
A third challenge is the increasing evidence that cloud services are being abused by

cyber-criminals, and that this trend is growing upwards [36]. Despite the recent efforts towards enhancing malware detection in cloud infrastructures, the use of dedicated malicious cloud services is generalizing and expanding. This motivates the need for improved security management services for cloud infrastructures.

1.2 Resilient multi-cloud network virtualization

With the above three challenges in mind – the dependency of a single cloud provider of current solutions; the lack of security and dependability techniques in their design; and the increasing number of security incidents involving the use of cloud services – in SUPERCLOUD we propose to develop a resilient multi-cloud network virtualization platform that spans multiple, heterogeneous Cloud Service Providers (CSPs). The solution will leverage on software-defined networking techniques to enable the creation of virtual networks that span multiple clouds in a transparent way to the SUPERCLOUD users. By such it will enable the SUPERCLOUD platform to offer Network-as-a-Service, supporting the specification and deployment of network components and their interconnecting channels.
Our solution addresses the first challenge of relying on a single cloud provider. The SUPERCLOUD network virtualization platform will leverage networking resources from multiple cloud providers and private clouds. This will allow distribution and replication of resources, enabling the platform to scale out to multiple clouds while tolerating single-cloud faults.
For the second challenge the platform design will include security and dependability principles and techniques from the outset, in both the network control plane and the data plane. Resilience will be built-in by design into the platform, ensuring that correct operation can be maintained under relevant failure scenarios, encompassing both problems of accidental and malicious natures. This includes the use of replication and distribution techniques to increase the dependability and scalability of SDN control, along with mechanisms for resilient communication and routing.
To address the final challenge the platform will include a self-managed network security framework. The use of SDN provides the flexibility required at the control plane to manage and configure the data plane, allowing the deployment of network detection, forensics and mitigation mechanisms for self-management.

1.3 Outline of the document

The reminder of this document is organized as follows. In Chapter 2 we specify the design requirements of the resilient multi-cloud network virtualization architecture. Then we present the state-of-the-art, with a short survey of Software-Defined Networking in Chapter 3, followed by a discussion of recently proposed SDN-based network virtualization platforms in Chapter 4. We motivate the need for improved security management services for the cloud in Chapter 5, by presenting a longitudinal study over the last six years on the nature and trends of cloud-based security incidents. Finally, we present the preliminary architecture in Chapter 6, with a focus on the various techniques proposed to fulfill the requirements of our design. Chapter 7 concludes this document.

Chapter 2 Design requirements

The design requirements of the SUPERCLOUD network architecture are influenced by both the target use cases and the cloud network infrastructure where they will be deployed. At this point, we anticipate two major use cases for the SUPERCLOUD platform: the SUPERCLOUD end users and the SUPERCLOUD providers. Figure 2.1 illustrates these use cases.

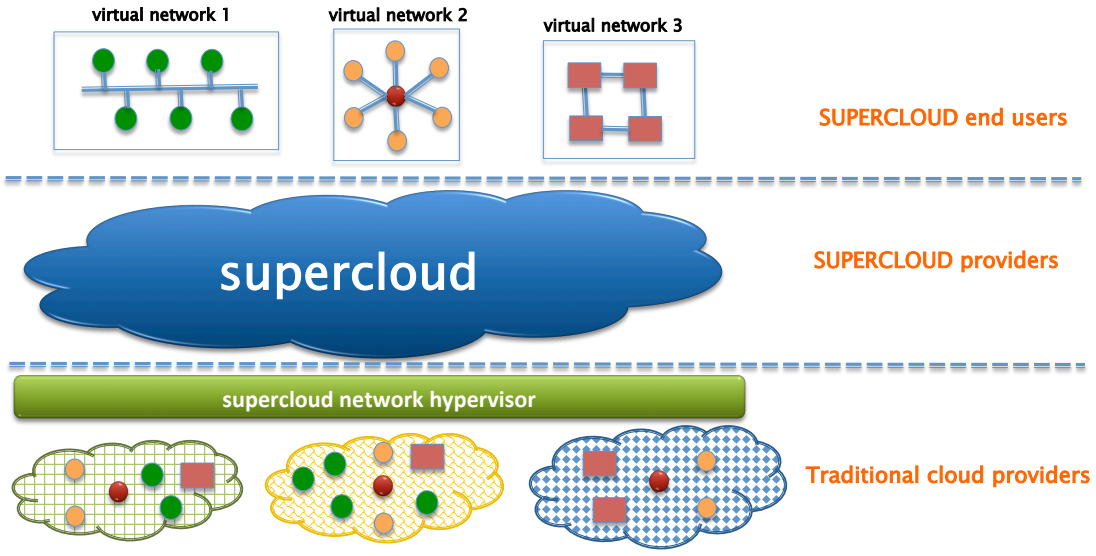


Figure 2.1: SUPERCLOUD target use cases: the SUPERCLOUD end users and the SUPERCLOUD providers.

SUPERCLOUD end users are organizations and companies that want to operate a network infrastructure across multiple clouds for different reasons, such as to scale out, for security and dependability, and/or due to legislation constraints. Each organization would run its own deployment of the SUPERCLOUD solution, which would be employed to specify the virtual networks and automatically and transparently have them instantiated in several clouds. In some scenarios, the end user may already own a datacenter (its own private cloud), which is then connected to facilities of Cloud Service Providers (CSPs) that may be physically located in different zones. The full set of resources can then be managed accordingly to the policies defined by the user, possibly differentiating the local facility from the remote ones.
SUPERCLOUD providers are companies (cloud operators or others) that want to supply virtual network services that run over multiple physical clouds. They use the SUPERCLOUD solution to program the cloud resources and offer virtual networks to their clients (or tenants) while maximizing efficiency and profit. These SUPERCLOUD providers may own a few facilities (their private clouds) and have agreements with other CSPs to extend and complement their offerings (e.g., to include a datacenter in a specific country). From the viewpoint of the clients of the company, they observe an aggregated cloud that can be employed as a whole to obtain the most appropriate service that

meets their needs. These clients no longer have to worry about multiple and usually inconsistent CSP Service Level Agreements (SLAs) and accounts, which simplifies the experience and the management processes. In this use case, one organization would have a single deployment of the SUPERCLOUD solution supporting multiple tenants that want to specify their individual network infrastructures.
These two use cases are generic and therefore they could be demonstrated with the healthcare applications addressed in the project, like the laboratory information system or the clinical image processing & storage (for more details see WP5). For instance, a company could function as a SUPERCLOUD end user by leveraging from its own datacenter to keep critical patient information that cannot leave its premisses (e.g., due to an agreement with a hospital), while offloading the rest of the data to multiple clouds. Since some of its clients (e.g., doctors and clinics) might be spread geographically, the company could place the data in the clouds that are nearer to the places where it is utilized for improved performance. Alternatively, a particular department of a company could act as a SUPERCLOUD provider for the other departments (or tenants). For example, the provider department would be responsible for maintaining agreements with several CSPs, offering to the rest of the company the aggregated multi-cloud as a service, in order to support for example different healthcare applications. SUPERCLOUD will leverage on network infrastructure from both public cloud providers (CSPs) and also private infrastructures (or private clouds) of the users (Figure 2.2). This heterogeneity impacts on the level of network visibility and control that may be achieved, affecting the type of configurations that can be pushed to the network, with obvious consequences on the kind of services and guarantees that can be assured by the SUPERCLOUD solution.

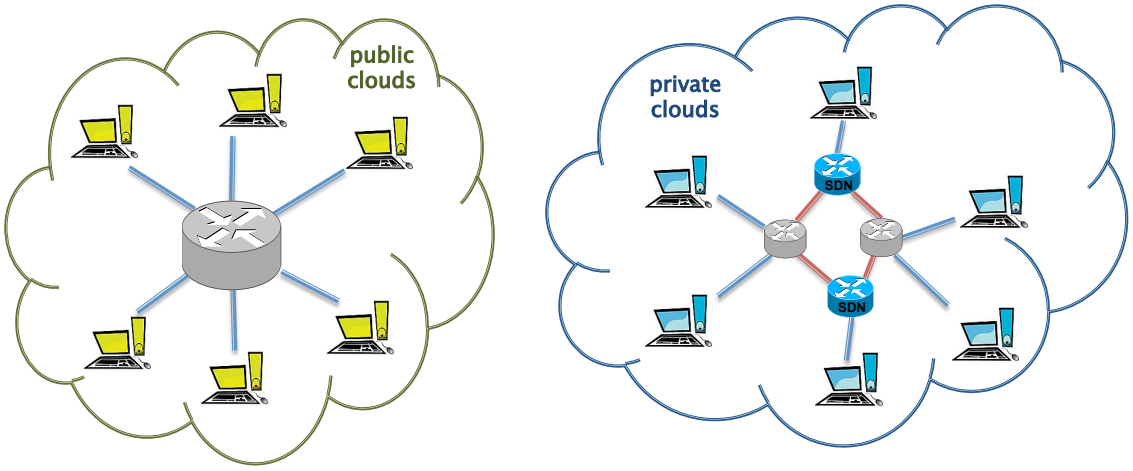


Figure 2.2: Infrastructure where the SUPERCLOUD solution will be deployed.

On one extreme case, the CSP gives very limited visibility and no (or extremely limited) network control, which is often the case with commercial cloud service providers (e.g., AWS). Even in this case, these clouds offer a full logical mesh among local VM instances (i.e., they provide a “big switch” abstraction), which we can use to implement logical software-defined datapaths and thus present a virtual network to the user. In this setting, one of the fundamental challenges that has to be addressed is how to interconnect the various CSPs, as this kind of support is normally unavailable, while ensuring levels of service compatible with the user requirements. In these clouds only software switching can be employed, as SUPERCLOUD does not have access to the hardware.
On the other extreme, full access may be attainable if the cloud is private (i.e., the datacenter belongs to the user). This results in a flexible topology that may be (partially) SDN-enabled, where both

software and hardware switching may be employed. This property is particularly interesting to build hybrid solutions. Hardware switches offer high-speed packet processing (hundreds of Gbps) but they have small rule tables (TCAMs are expensive). On the other hand, software switches do not forward packets at high speeds (10/20 Gbps per core maximum), but have a large rule space.
Considering this setting, the multi-cloud network virtualization platform has the following requirements:

- Need for remote, flexible control of network elements. This is the enabler for the solution.
- Security and dependability of both the control and data planes.
- Full network virtualization, including topology abstraction, addressing abstraction, and isolation between tenants.
- Autonomic security management.
- Ability to perform a network snapshot.
- Scalability and performance.
- Compatibility and interoperability.

In the following sections we further detail each of these requirements.

2.1 Remote, flexible control of the network

Users of SUPERCLOUD define virtual networks which are then deployed on the clouds of various providers (and eventually on its own datacenter). In order to connect the actual elements, SUPERCLOUD needs to be able to control the operation (namely, the packet forwarding rules) of the network devices to establish the channels that carry the traffic exchanged among the VMs. In addition, it is also necessary to devise mechanisms that allow the creation of links that cross facilities, as typical cloud offerings do not provide such support.

- **DR1 Network Controllability** SUPERCLOUD can control the behavior of network devices to setup the connections to the various elements of the infrastructure, ensuring the necessary levels of quality of service.

The project will resort to the SDN paradigm [47] to build the solution. The SUPERCLOUD virtualization layer will include a network virtual switch whose operation can be remotely updated by a controller. A standardized protocol, such as Openflow [53], will be used to orchestrate the exchanges between the controller and the switches. This protocol is implemented by the most common SDN-based physical and virtual switches, enabling a reasonable level of programmability of the network.

2.2 Security and dependability

- **DR2 Network Availability** Users should observe an available network, where communications can occur with high probability.
- **DR3 Network Security** Users should experience a secure network, where appropriate measures are applied to protect the communications from attacks.

The project aims to allow the instantiation of VMs and virtual networks among public cloud and private facilities, which will have to be interconnected over the Internet. Therefore, there are multiple issues related to dependability and security that need to be solved. Solutions for these issues typically

involve tradeoffs, namely with regard to performance and cost (e.g., a higher availability requires a greater level of link redundancy, which normally is more expensive to maintain).
On the data plane there is the need to deploy mechanisms to ensure packet delivery under different load and failure conditions to guarantee the availability of the network. Alternative approaches, such as multipath routing [8, 54, 29] and network coding [43], can contribute for an improvement on the way the network copes with potential failures, but they may have their own drawbacks which need to be further studied.
In addition, communications have to be secured and monitored to prevent successful attacks. For instance, higher-quality paths are assigned to legitimate traffic, while lower-quality paths are employed to reroute the part of the traffic that is perceived as malicious (e.g., to a middle-box where it can be further analyzed with an intrusion detection system). Of course, in the general case, it might be impossible to distinguish good from bad packets, but for many kinds of attacks this approach can be effective. In this case, the network security policies would need to be defined to describe which actions should be performed based on the context of the network. The specification of these security policies may be realized using an access control formalism (such as RBAC [65], ABAC [77] or OrBAC [7]) in order to design complete policies where conflicts can be rapidly detected, define priorities and simulate these policies. Finally, the communications amongst public cloud providers and private infrastructure may require a lightweight authentication system as well as a potential standardization for the messages exchanged. Message signature and non-repudiation mechanisms might also be considered.
In SDN technologies, the controller is logically centralized, and therefore it can become a bottleneck in data processing or a single point of failure. In addition, it can be subject to different forms of attacks, such as Distributed Denial of Service (DDoS), which can impact on network management. The use of replication techniques coupled with the diversity on the controller platform could make the control plane more resilient.

2.3 Full network virtualization

- **DR4 Network Virtualization** Users define an arbitrary virtual network topology unaware of other tenants.

SUPERCLOUD should enable the specification of an arbitrary virtual network topology, which is independent of the existing physical infrastructure. Alternative mappings can then be utilized when the network is actually setup. For instance, a virtual link can correspond to multiple network paths connecting the two endpoints. Virtual switches may hide large portions of the infrastructure, where several (physical) switches have to be interconnected in order to provide the required abstraction.
In addition, users should have complete autonomy to manage their own address space of the virtual network. Address virtualization can be employed to achieve this goal, allowing users to assign unrestrained address identifiers to the virtual communication endpoints. These addresses are mapped to the actual physical addresses (e.g., specific IP addresses) avoiding conflicts. In other words, even if two users unknowingly give the same virtual address to some of their VMs, these addresses are translated to distinct physical addresses after the mapping.
Lastly, isolation between users could be enforced at different degrees. A first level is attained by separating the virtual networks of the users and then hiding them from each other when they are deployed. A second level is to prevent the actions of one user to influence the network behavior observed by the others. For example, if one of the users attempts to clog a particular link, this should not cause a significant decrease on the bandwidth available to the other users.

2.4 Autonomic security management

- **DR5 Autonomic Security Management** SUPERCLOUD should be able to monitor and react to security incidents automatically.

SUPERCLOUD will leverage on heterogeneous network infrastructure from both public cloud providers and private infrastructures. Each provider (both public providers and private infrastructures) would have only a partial view of security incidents that may affect the SUPERCLOUD users. For example, an attack that leverages SUPERCLOUD resources may share the same artifacts across multiple heterogeneous providers. Therefore, it would be easier and more efficient to characterize this attack when correlating observations across all these providers. Besides, a comprehensive approach for attack remediation may require to coordinate actions such as VM quarantine or network migration across multiple cloud providers. Note that security incidents may be manifested in two different ways. On the first hand, SUPERCLOUD users may be affected by cyberattacks such as DDoS and malware infections. On the second hand, security incidents also include abuses of SUPERCLOUD resources that may occur when malicious SUPERCLOUD users resort to the heterogeneous multi-cloud environment to coordinate or to launch attacks against third parties.
The autonomic security management framework aims to address these challenges. It offers data plane monitoring and proactive attack detection across the different providers of the SUPERCLOUD. It enables to characterize and classify security incidents that may occur within the SUPERCLOUD, and implement tunable reaction policies that would segregate and isolate attack traffic.

2.5 Network snapshot

- **DR6 Network Snapshot** Users should be able to take a snapshot of the virtual network and later on restart it again.

After a virtual infrastructure is deployed and has been running for a while, the user might want to stop it and then restart it later on (e.g., to improve dependability or to minimize costs, if cloud resources have different prices during the day). A fundamental service to achieve this goal is the ability to snapshot a VM at a particular instant. For example, a facility failure could be tolerated with the creation of a VM snapshot, which is then stored at another location — this allows the VM to be restarted at a different place, hopefully not affected by the problem. In the worst case, some recent tasks performed by the VM after the snapshot would be lost, but the computation would not need to be initiated from the beginning.
In order to offer VM snapshot creation, SUPERCLOUD needs to capture the network state relevant to the VM and then update it after the restart. This normally means collecting flow rules (and other configurations) that are stored in the virtual switch that connects the VM, but also forwarding rules that are spread through the data plane and that are related to the VM communications. The network embedding algorithms, used to define the mapping between the physical and virtual entities, may also need to be rerun to ensure that after the new physical location of the VM is picked, the network is still able to accommodate the expected traffic to be exchanged among VMs and also with the external nodes.

2.6 Scalability

- **DR7 Network Scalability** Users can deploy arbitrarily sized virtual networks without relevant performance degradation.

The network virtualization solution should not introduce constrains on the number of VMs, network devices and amount of traffic in the virtual infrastructure, as long as enough (physical) resources are made available to support them. This means that, for instance, the user can specify large groups of VMs, and then they are automatically instantiated in the clouds with the necessary links to connect them. Since the calculation of the optimal mappings from virtual to physical components can take a while when the number of elements is high, SUPERCLOUD should be able to operate with non-optimal (but still acceptable) solutions to the problem.

SUPERCLOUD should not introduce unnecessary overheads in the physical network. Ideally, the user should experience a quality of service on the delivery of traffic that is similar to the offering that was contracted with the CSPs. However, one should keep in mind that packets crossing the border of a datacenter will suffer a non-negligible delay when compared to internal communications. Moreover, packets flowing from one CSP to another will be even more delayed as they will have to go through the Internet (since they cannot travel through the private links that typically connect the datacenters of the same provider).

2.7 Compatibility and interoperability

- **DR8 Network Interoperability** Users can deploy virtual networks over most of the existing cloud service providers.

SUPERCLOUD will develop a solution that operates over several CSPs. Currently, CSPs offer heterogeneous solutions when it comes to the virtualization platform and the type of services that are made available to the users. Therefore, in order to support the above requirements, SUPERCLOUD needs on one side to abstract the inconsistencies of the CSPs services, giving a common view of the existing resources and simplifying their utilization. On the other side, the technological solutions that are implemented for the network virtualization have to be highly portable, ensuring that they can be used over a high variety of potential CSPs. One aspect that still needs to be better studied is how to create good level of abstraction while still making available to the user specific features of a particular cloud (e.g., certain aspects of private clouds), without increasing significantly the setup effort.

Chapter 3 A primer on Software-Defined Networking

Traditional computer networks are *complex and very hard to manage* [12]. To express the desired policies, network operators need to configure each individual network device, one by one, either manually or with the use of low-level scripts. In addition to configuration complexity, network environments have to endure the dynamics of faults and adapt to load changes. Enforcing the required policies in such a dynamic environment is highly challenging. Current networks are also *vertically integrated*. The control plane (that decides how to handle network traffic) and the data plane (that forwards traffic according to the decisions made by the control plane) are bundled inside the networking devices. This is a fundamental obstacle that has led to the slow pace of innovation of networking infrastructure.
Software-Defined Networking (SDN) [47] is an emerging paradigm that promises to change the current state of affairs. SDN breaks the vertical integration by separating the network's control logic from the underlying routers and switches that forward the traffic. Network switches become simple forwarding devices and the control logic is implemented in a *logically centralized* controller, simplifying policy enforcement. While the controller can be implemented as a distributed system, network applications have access to a centralized programmatic model (a global network view), making it easier to reason about network behavior. A simplified view of this architecture is shown in Figure 3.1.

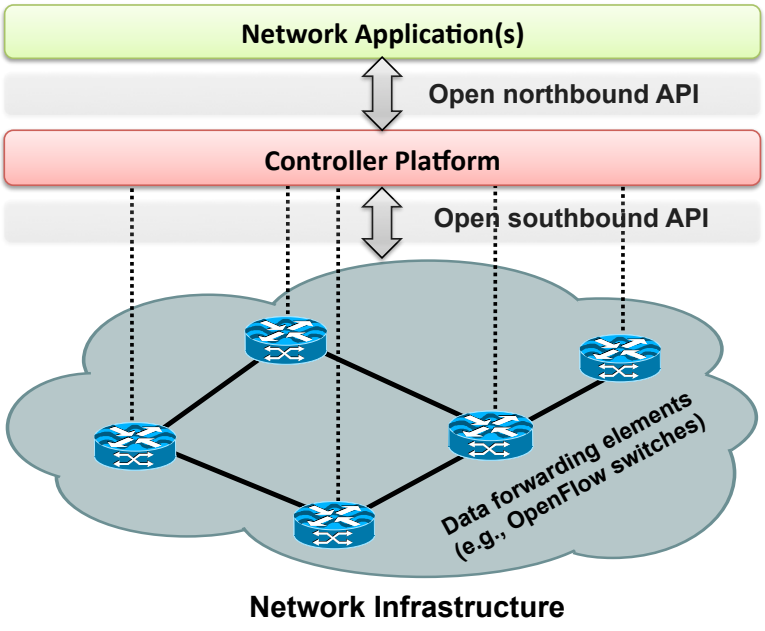


Figure 3.1: Simplified view of an SDN

The separation of the control plane and the data plane can be realized by means of a well-defined programming interface between the switches and the SDN controller. The controller exercises direct control over the state in the data-plane elements via this well-defined southbound interface. The most notable example of such interface is OpenFlow [53]. An OpenFlow switch has one or more tables of packet-handling rules. Each rule matches a subset of the traffic and performs certain actions on the packets (dropping, forwarding to specific port(s), modifying headers, among others). Depending on

the rules installed by a control application, an OpenFlow switch can – instructed by the controller – behave like a router, switch, firewall, load balancer, or perform other roles.
An important consequence of the software-defined networking principles is the *separation of concerns* introduced between the *definition* of network policies, their *implementation* in switching hardware, and the *forwarding* of traffic. This separation allows modularity by breaking the network control problem into tractable pieces, and making it easier to create and introduce new abstractions in networking [66]. This is key to simplify network management and to facilitate innovation.
Although SDN and OpenFlow started as an academic experiment [53], they gained significant traction in the industry over the past few years. Most vendors of commercial switches now include OpenFlow support in their equipment. The ideas behind SDN have matured and evolved from an academic exercise to a commercial success. The world's largest IT companies have recently joined SDN consortia such as the Open Networking Foundation (ONF) [5] as an indication of the importance of SDN from an industrial perspective.
In this chapter we introduce Software-Defined Networking and show how this paradigm differentiates from traditional networking (Section 3.1). We also present the most relevant building blocks of the SDN infrastructure (Section 3.2). We conclude the chapter with a description of the fundamental security and dependability challenges of this architecture [63].

3.1 Software-Defined Networking

The term SDN was originally coined to represent the ideas and work around OpenFlow at Stanford University [32]. A software-defined network is a network architecture with four pillars:

1. The control and data planes are *decoupled*. Control functionality is removed from network devices that become simple (packet) forwarding elements.

2. Forwarding decisions are flow-based, instead of destination-based. A flow is broadly defined by a set of packet field values acting as a match (filter) criterion and a set of actions (instructions). The flow abstraction allows unifying the behavior of different types of network devices, including routers, switches, firewalls, and other middleboxes.

3. Control logic is moved to an external entity, the SDN controller. The controller is a software platform that runs on commodity server technology and provides the essential resources and abstractions to facilitate the programming of forwarding devices based on a logically centralized, abstract network view. Its purpose is therefore similar to that of a traditional operating system, but for networking resources.

4. The network is *programmable* through software applications running on top of the SDN controller. This is a fundamental characteristic of SDN, considered its main value proposition.

Following the concepts introduced in [66], an SDN can be defined by three fundamental abstractions: (*i*) forwarding, (*ii*) distribution, and (*iii*) specification. Ideally, the *forwarding abstraction* should allow any forwarding behavior desired by the network application (the control program) while hiding details of the underlying hardware. OpenFlow is one realization of such abstraction, which can be seen as the equivalent to a "device driver" in an operating system. The *distribution abstraction* should shield SDN applications from the vagaries of distributed state, logically centralizing the network control, guaranteeing its consistency. Its realization requires a common distribution layer, which in SDN resides in the controller. The last abstraction is *specification*, which should allow a network application to express the desired network behavior without being responsible for implementing that behavior itself.

3.2 SDN building blocks

An SDN architecture can be depicted as a composition of different layers, as shown in Figure 3.2. Each layer has its own specific function.

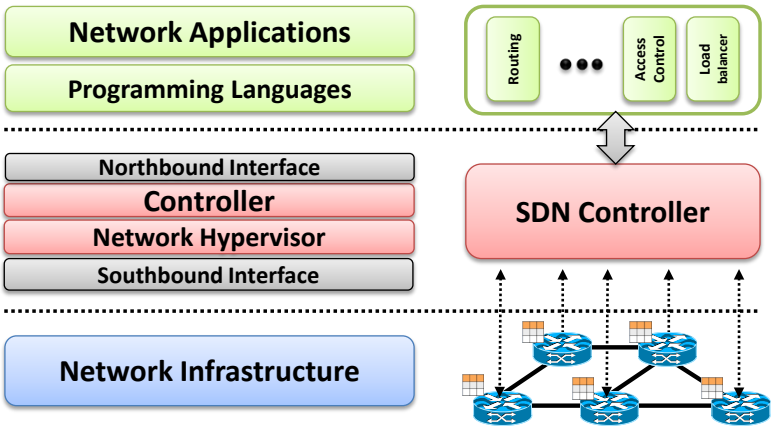


Figure 3.2: A layered view of SDN

3.2.1 Network infrastructure

An SDN infrastructure, similarly to a traditional network, is composed of a set of networking equipment (switches, routers, and middlebox appliances). The main difference resides in the fact that those traditional physical devices are now simple forwarding elements without (or with limited) embedded control or software to take autonomous decisions. The network intelligence is removed from the data plane devices to a logically-centralized controller. In most instances of SDN this separation of the control and data planes is materialized by means of the OpenFlow protocol [53]. Openflow is the communication protocol that allows the controller to centrally control the behavior of the network switches (more details in the next section). An OpenFlow-enabled forwarding device is based on a pipeline of flow tables where each entry of a flow table has three parts:

1. a matching rule,
2. actions to be executed on matching packets, and
3. counters that keep statistics of matching packets.

Inside an OpenFlow device, a path through a sequence of flow tables defines how packets should be handled. When a new packet arrives, the lookup process starts in the first table and ends either with a match in one of the tables of the pipeline or with a miss (when no rule is found for that packet). A flow rule can be defined by combining different matching fields. If there is no default rule, the packet will be discarded. However, the common case is to install a default rule which tells the switch to send the packet to the controller (or to the normal non-OpenFlow pipeline of the switch). Possible actions thus include:

1. forward the packet to outgoing port(s),
2. encapsulate it and forward it to the controller,
3. drop it,
4. send it to the normal processing pipeline,
5. send it to the next flow table or to special tables.

3.2.1.1 Open vSwitch

Several OpenFlow-enabled forwarding devices are available on the market, both as commercial and open source products. Networking hardware manufacturers have produced various kinds of OpenFlow-enabled devices. These devices range from equipment for small businesses (e.g., GbE switches) to high-class data center equipment.
Software switches are emerging as one of the most promising solutions for data centers and virtualized network infrastructures [67, 18]. They have been used for moving network functions to the edge (with the core performing traditional IP forwarding), thus enabling network virtualization [44]. The most notable example of a software-based OpenFlow switch implementation is Open vSwitch (OvS) [60]. OvS is a software switch that operates within the hypervisor/management domain and provides connectivity between the virtual machines and the underlying physical interfaces. It implements standard Ethernet switching and in a standalone configuration, it operates much like a basic L2 switch. However, to support integration into virtual environments, and to allow (logical) switch distribution, OvS exports interfaces for manipulating the forwarding state and managing configuration state at runtime. Underlying OvS there is a flow-table forwarding model similar to that used by OpenFlow. Since OpenFlow does not allow to modify the switch configurations (e.g. configure queues and tunnels, add/remove ports, create/destroy switches), OvS also maintains a database and exports a configuration interface that enables remote configuration of the virtual switches via the OVSDB protocol [59].
Since OvS was originally developed its performance was on par with the Linux Ethernet bridge. Over the past few years its performance has been gradually optimized to match the requirements of multi-tenant datacenter workloads [60].

3.2.2 Southbound interfaces

Southbound interfaces are the connecting bridges between control and forwarding elements, thus being the crucial instrument for clearly separating control and data plane functionality. OpenFlow is the most widely deployed open southbound standard for SDN. It provides a common specification to implement OpenFlow-enabled forwarding devices, and for the communication channel between data and control plane devices. The OpenFlow protocol provides three information sources for controllers. First, event-based messages are sent by forwarding devices to the controller when a link or port change is triggered. Second, flow statistics are generated by the forwarding devices and collected by the controller. Third, packet-in messages are sent by forwarding devices to the controller when they do not known what to do with an incoming packet or because there is an explicit "send to controller" action in the matched entry of the flow table. These information channels are the essential means to provide flow-level information to the controller.
As explained above, OVSDB [59] is another type of southbound API, designed to provide advanced management capabilities for Open vSwitches [60]. Beyond OpenFlow's capabilities to configure flows and thus influence forwarding behavior, OVSDB allows the creation of tunnels, turning on or off certain features, get configuration data, etc. OVSDB is therefore a complementary protocol to OpenFlow for Open vSwitch.

3.2.3 Network hypervisor

Long standing virtualization primitives such as VLANs, NAT, and MPLS provide only limited forms of network virtualization. These solutions are also anchored on a box-by-box basis configuration. The programmability offered by an SDN has the means to provide full network virtualization – not only isolation of virtual networks, but also topology and addressing virtualization.
FlowVisor [69] was the earliest attempt to virtualize an SDN. This network hypervisor acts as a proxy between the controller and the forwarding devices to provide an abstraction layer that slices an OpenFlow data plane, allowing multiple controllers each to control its share (its slice) of a single physical infrastructure. The isolation properties provided by FlowVisor thus allow several networks to co-exist. OpenVirteX [9] extends FlowVisor to provide not only isolation of network control,

but also topology and address virtualization. Contrary to these hypervisors that virtualize an SDN, VMware's Network Virtualization Platform (described in more detail in the next chapter) provides full virtualization in data centers without requiring SDN-based hardware (the only requirement is that all servers are virtualized).

3.2.4 Controller

The controller is the fundamental element in an SDN architecture, as it is the key supporting piece for the control logic (applications) to generate the network configuration based on the policies defined by the network operator. Similar to a traditional operating system, the control platform abstracts the lower-level details of the interaction with forwarding devices.
There is a diverse set of controllers with different design and architectural choices. Existing controllers can be categorized based on many aspects. From an architectural point of view, one of the most relevant is if they are centralized or distributed. A centralized controller (such as NOX [34] and Floodlight [3]) is a single entity that manages all forwarding devices of the network. Naturally, it represents a single point of failure and may have scaling limitations. Contrary to a centralized design, a distributed controller (such as Onix [45]) can be scaled up to meet the requirements of potentially any environment, from small to large-scale networks.

3.2.5 Northbound interface

The north and southbound interfaces are two key abstractions of the SDN ecosystem. The southbound interface has already a widely accepted proposal (OpenFlow), but a common northbound interface is still an open issue. At this moment it may still be a bit too early to define a standard northbound interface, as use-cases are still being worked out [24]. Anyway, it is to be expected a common (or a *de facto*) northbound interface to arise as SDN evolves.

3.2.6 Programming languages

OpenFlow resembles an assembly-like machine language, essentially mimicking the behavior of forwarding devices, forcing developers to spend too much time on low-level details rather than on the problem to solve. Raw OpenFlow programs have to deal with hardware behavior details such as overlapping rules, the priority ordering of rules, and data-plane inconsistencies that arise from in-flight packets whose flow rules are under installation. The use of these low-level languages makes it difficult to reuse software, to create modular and extensive code, and leads to a more error-prone development process. Abstractions provided by high level network programming languages [30] can significantly help address many of the challenges of these lower-level instruction sets.

3.2.7 Network applications

Network applications can be seen as the "network brains". They implement the control-logic that will be translated into commands to be installed in the data plane, dictating the behavior of the forwarding devices. Take a simple application as routing as an example. The logic of this application is to define the path through which packets will flow from a point A to a point B. To achieve this goal a routing application has to, based on the topology input, decide on the path to use and instruct the controller to install the respective forwarding rules in all forwarding devices on the chosen path, from A to B.
Existing SDN applications either perform traditional functionality such as routing, load balancing, and security policy enforcement, or explore novel approaches, such as reducing power consumption [37]. Other examples include fail-over and reliability functionalities to the data plane, end-to-end QoS enforcement, network virtualization, mobility management in wireless networks, among many others [47]. The variety of network applications, combined with real use case deployments, is expected to be one of the major forces on fostering a broad adoption of SDN.

3.3 Security and dependability

SDN provides new ways to solve age-old problems in networking — routing [17], for instance — , while simultaneously enabling the introduction of sophisticated network policies, such as security and dependability. Take Ethane [19], for example, an SDN architecture that allows managers to enforce fine-grained access control policies. However, the security and dependability of the SDN *itself* has been a relatively neglected topic until recently [46].

With the growing number of SDN deployments [40, 39, 48] security and dependability issues are becoming a concern for the industry. Past research on SDN security [46] has identified several potential threat vectors in SDNs. The main conclusion of such studies is that SDN poses threats of a different nature of traditional networking, and hence need to be dealt differently.

Interestingly, the main causes of concern lie in SDN main benefits: network programmability and control logic centralization. These capabilities open the doors for new threats that did not exist before. Traditional networks have "natural protections" against network attacks. Namely, the closed (proprietary) nature of network devices, the heterogeneity of software, and the distributed nature of the control plane represent defenses against common vulnerabilities.

Software-defined networks have two properties which can be seen as attractive for malicious users. First, the ability to control the network by means of software. Second, the centralization of the "network brains" in the controller(s). Anyone with access to the servers that host the control software can potentially control the entire network.

Despite this modification of the attack surface, software-defined networks are not inherently less secure than current networks. However, SDNs pose threats of a different nature that need therefore to be dealt with differently. On the contrary, if the SDN is properly designed and deployed, this new network environment offers the agility needed to quickly deploy new changes to the control and data planes, and use some advanced capabilities like model checking or proactive security, which are often not possible with traditional mechanisms.

These challenges motivate the need to consider security and dependability as first class properties when designing an SDN. That is therefore one of the fundamental requirements to fulfill in the design of the SUPERCLOUD network virtualization platform.

Chapter 4 SDN-based network virtualization: state-of-the art

Hypervisors enable distinct virtual machines to share the same hardware resources. In a cloud infrastructure-as-a-service (IaaS), each user can have its own virtual resources, from computing to storage. This technology enabled new revenue and business models where users allocate resources on-demand, from a shared physical infrastructure, at a relatively low cost. At the same time, providers make better use of the capacity of their installed physical infrastructures, creating new revenue streams without significantly increasing their capital expenditure (CAPEX) and operational expenditure (OPEX) costs.
Virtualization is already a consolidated technology in modern computers. The fast developments of the past decade have made virtualization of computing platforms mainstream. Based on recent reports, the number of virtual servers has already exceeded the number of physical servers [13, 44].
One of the interesting features of virtualization technologies today is the fact that virtual machines can be easily migrated from one physical server to another and can be created and/or destroyed on-demand, enabling the provisioning of elastic services with flexible and easy management. Unfortunately, virtualization has been only partially realized in practice. Despite the great advances in virtualizing computing and storage elements, the network is still mostly statically configured in a box-by-box manner [22].

4.1 Network requirements

The main network requirements can be captured along two dimensions: network topology and address space. Different workloads require different network topologies and services, such as flat L2 (Layer 2) or L3 (Layer 3) services, or even more complex L4-L7 services for advanced functionality. Currently, it is very difficult for a single physical topology to support the diverse demands of applications and services. Similarly, address space is hard to change in current networks. Nowadays, virtualized workloads have to operate in the same address of the physical infrastructure [44]. Therefore, it is hard to keep the original network configuration for a tenant, virtual machines can not migrate to arbitrary locations, and the addressing scheme is fixed and hard to change. For example, IPv6 cannot be used by the virtual machines (VMs) of a tenant if the underlying physical forwarding devices support only IPv4.
To provide complete virtualization the network should provide similar properties to the computing layer [22]. The network infrastructure should be able to support arbitrary network topologies and addressing schemes. Each tenant should have the ability to configure both the computing nodes and the network simultaneously.
One might think that long standing virtualization primitives such as VLANs (virtualized L2 domain), NAT (virtualized IP address space), and MPLS (virtualized path) are enough to provide full and automated network virtualization. However, these technologies are anchored on a box-by-box basis configuration, i.e., there is no single unifying abstraction that can be leveraged to configure (or re-configure) the network in a global manner. As a consequence, current network provisioning can take months, while computing provisioning takes only minutes [44, 21, 55, 78].
In the past few years this situation has began to change with SDN and the availability of new tunneling techniques. A number of network virtualization solutions have been proposed, mostly targeting a single

multi-tenant datacenter. In this chapter we survey this recent work.

4.2 Layer 2 (L2) tunneling

The widespread use of Ethernet technology and the worldwide scale of today's cloud infrastructures [40] has recently created the need for these networks to be extended globally. For this purpose, multi-tenant network virtualization platforms [44] rely on tunneling techniques to implement the needed logical communication between hosts. In this solution, when a packet is generated in one VM, the tenant's traffic needs to be tunneled across the physical network to the receiving host hypervisor for delivery to the destination VM.
The primary goal of the tunneling techniques currently used for network virtualization was to extend Local Area Networks (LANs). Contrary to IP addresses, Ethernet addresses do not change as hosts move, and hence their use as host ID makes it very easy to move systems in the datacenter. Unfortunately, extending a LAN to make full use of this property is faced with numerous challenges. First, LANs make extensive use of broadcast to discover unknown hosts, and the excessive floodling brings with it scalability issues (e.g., creating broadcast storms). Second, the Spanning Tree Protocol (STP) that is used to avoid loops in these networks creates problems as the spanning tree root can easily become a bottleneck and is a single point of failure. When things go wrong even transient loops are hard to avoid, as techniques used in IP networks such as hop count do not exist in LANs. Finally, STP-based routing is inefficient, as it is centered in the root and does not allow the use of multiple paths. As a consequence, even while some links remain unused others may be fully congested.
In this section we look at encapsulation (tunneling) techniques that have been recently introduced to allow Ethernet to span over Layer-3 (L3) networks, allowing multi-tenancy in data centers.

4.2.1 TRILL

IP subnets are not good for mobility because as a host moves its IP addresses needs to change, breaking connectivity. By contrast, LANs allow free mobility inside the LAN, as MAC addresses do not change as a host moves, allowing location-independent addressing. However, as explained above, LANs face numerous scalability and efficiency problems. On the contrary, IP routing is scalable and efficient.
The TRILL (Transparent Interconnection of Lots of Links) IETF standard [58] was proposed to allow a large campus to become a single extended LAN, allowing people to move around. The goal was to take advantage of the best of L2 switching and L3 routing. MAC addressing is used to allow for mobility, while IP routing is used for efficiency.
In TRILL a special type of switches, the Routing Bridges (RBridges) [57], encapsulate L2 frames and route them to destination RBridges, which decapsulate them and forward to destination. The encapsulation is shown in Figure 4.1, with Ethernet frames encapsulated in a TRILL header, which is further encapsulated using an outer header (e.g., IP).

Figure 4.1: Encapsulation formats used for network virtualization

RBridges learn MAC addresses by source learning and by exchanging their MAC tables with other RBridges. TRILL switches run a routing link state protocol (IS-IS) amongst themselves to exchange

MAC tables. In such way RBridges obtain the necessary information to compute pair-wise optimal paths. To mitigate temporary loop issues, RBridges makes use of another IP technique: forwarding is based on a header that includes a hop count. RBridges could be connected by any type of network equipment – so the outer header could be any protocol (e.g., IP, MPLS, etc.).

4.2.2 NVGRE

GRE, Generic Routing Encapsulation [26], is a generic encapsulation method that allows any protocol to run over any other protocol. Similar to TRILL, Network Virtualization using GRE (NVGRE) [31] uses Ethernet over GRE for L2 connectivity (Figure 4.1). Typically, IP or IPSec are used as delivery headers. NVGRE scales to multi-tenant scenarios, allowing 2^{24} tenants (over 16 million) to share the infrastructure (in contrast to VLANs, that constrains the number of tenants to 4096). In addition, Equal Cost Multipath (ECMP) is allowed on point-to-point tunnels, increasing network efficiency.

4.2.3 VXLAN

Virtual eXtensible Local Area Networks (VXLAN) [52] is another L3 solution used to isolate multiple tenants in a data center. It was developed by VMware and is supported by many companies in the IETF NVO3 working group. It is for this reason the most common protocol in datacenters for network virtualization and has support from all the major switch hardware vendors. VXLAN is very similar to NVGRE: both solutions offer basically the same functionality. The main difference is the fact that VXLAN encapsulates Ethernet frames over UDP over IP, instead of GRE (Figure 4.1).

4.2.4 STT

The Stateless Transport Tunneling Protocol (STT) [23] was developed by Nicira, the SDN-based company acquired by VMware to build this company's multi-tenant network virtualization platform [44]. STT uses Ethernet over a TCP-like stateless protocol over IP (Figure 4.1). The outer protocol is said TCP-like because it is a stateless version of TCP, not including common TCP mechanisms such as retransmissions or congestion control (it does keep the same protocol number, however).
STT solves the problem of efficient transport of large storage blocks in datacenters. The typical block storage size in datacenters is 64kB, with an Ethernet frame carrying only 1.5kB (jumbo frames extend this value to 9kB of payload). One of the features of STT is that it was designed to handle large storage blocks of 64kB.
In addition, STT re-enables hardware offloading for encapsulated traffic with existing Network Interface Cards (NICs). Current Ethernet NICs do not support offloading in the presence of IP encapsulation in the packet. As STT places a standard TCP header after the physical IP header, it takes advantage of the NICs that perform hardware offloading, namely LSO (Large Send Offload) and LRO (Large Receive Offload). LSO allows the operating system to send TCP packets larger than the physical MTU (Maximum Transmission Unit) to a NIC, which then splits them and computes the TCP checksums for each packet, on behalf of the OS. LRO does the opposite, aggregating multiple incoming packets into a single large TCP packet and, after verifying the checksum, handing it over to the Operating System (OS). The combination of these mechanisms provides a significant reduction in CPU usage for high-volume TCP transfers. Enabling these techniques is therefore an important advantage of STT.

4.2.5 Geneve

Despite its advantages, the standardization of STT has been discontinued recently. To fill this gap, currently the IETF is working on a VXLAN extension called Geneve [33], expected to become the future reference. The objective of this protocol is to combine the best of current network virtualization encapsulations (VXLAN, NVGRE, and STT) into a single protocol.

One singular feature of Geneve is its flexible and extensible tunnel format, which enables rich capabilities for network virtualization. As an example, metadata is encoded in an extensible TLV (type-length-value) structure within Geneve. Also, like NVGRE and VXLAN, the switches can continue to distribute traffic across links using ECMP based on the outer packet header fields. As in STT, support for NICs that provide stateless offloads is also included.

4.3 Slicing the network

FlowVisor [69] was the seminal work on virtualizing an SDN. Its basic idea is to allow multiple logical networks share the same OpenFlow networking infrastructure. For this purpose, it provides an abstraction layer that makes it easier to slice a data plane based on off-the-shelf OpenFlow-enabled switches, allowing multiple and diverse networks to co-exist.
FlowVisor sits between the tenants controllers and the SDN network switches (Figure 4.2). From a system design perspective, FlowVisor is a transparent proxy that intercepts OpenFlow messages between switches and controllers. By such, FlowVisor controls the view that the tenants controllers have of the SDN switches.

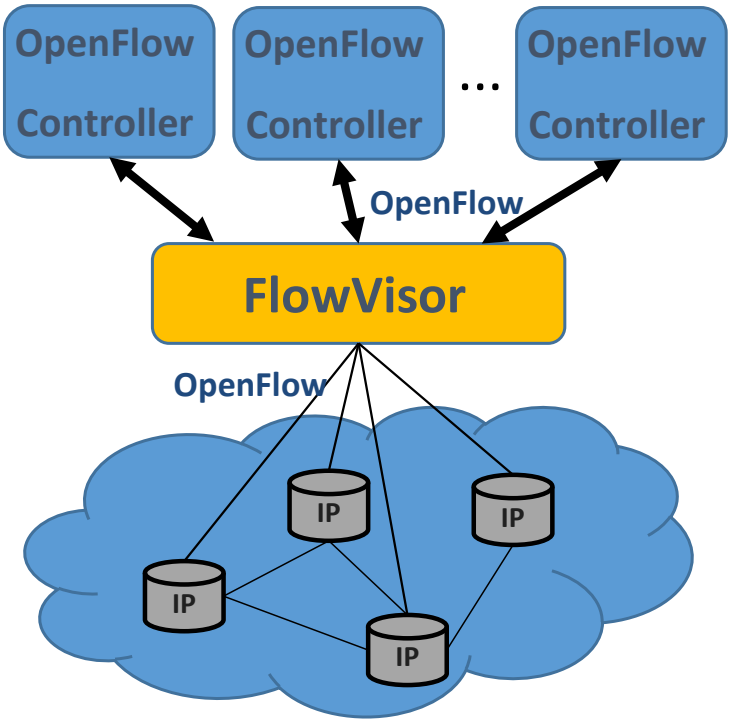


Figure 4.2: FlowVisor architecture

Five slicing dimensions are considered in FlowVisor: bandwidth, topology, traffic, device CPU, and forwarding tables. Different mechanisms are used to slice over each dimension. For instance, VLAN priority bits are used for bandwidth isolation. The proxy-based architecture allows packets to be intercepted – both from and to the controller – allowing transparent slicing of topology and forwarding tables.
In FlowVisor each network slice supports a controller, i.e., multiple SDN controllers can co-exist on top of the same physical network infrastructure. Each controller is allowed to act only on its own network slice. In general terms, a slice is defined as a particular set of flows on the data plane. Each slice receives a minimum data rate and each guest controller gets its own virtual flow table in the switches.
Other slicing approaches based on FlowVisor appeared in the literature recently. For instance, Au-

toSlice [16] focuses on the automation of the deployment and operation of vSDN (virtual SDN) topologies with minimal mediation or arbitration by the substrate network operator. Additionally, AutoSlice targets scalability aspects of network hypervisors by optimizing resource utilization and by mitigating the flow-table limitations through a precise monitoring of the flow traffic statistics. Similarly, AutoVFlow [74] also enables multi-domain network virtualization. However, instead of having a single third party to control the mapping of vSDN topologies, as is the case of AutoSlice, AutoVFlow uses a multi-proxy architecture that allows network owners to implement flow space virtualization in an autonomous way by exchanging information among the different domains.

4.4 Full network virtualization

FlowVisor slicing approach does not offer full network virtualization. It merely slices the network to be shared among multiple tenants. OpenVirteX [9] (OVX) builds on the design of FlowVisor, acting as a proxy between the controller and the forwarding devices (Figure 4.3), while providing virtual SDNs through both topology, address, and control function virtualization. All these properties are necessary in multi-tenant environments where virtual networks need to be managed and migrated according to the computing and storage virtual resources. Virtual network topologies have to be mapped onto the underlying forwarding devices, with virtual addresses allowing tenants to completely manage their address space without depending on the underlying network elements addressing schemes.

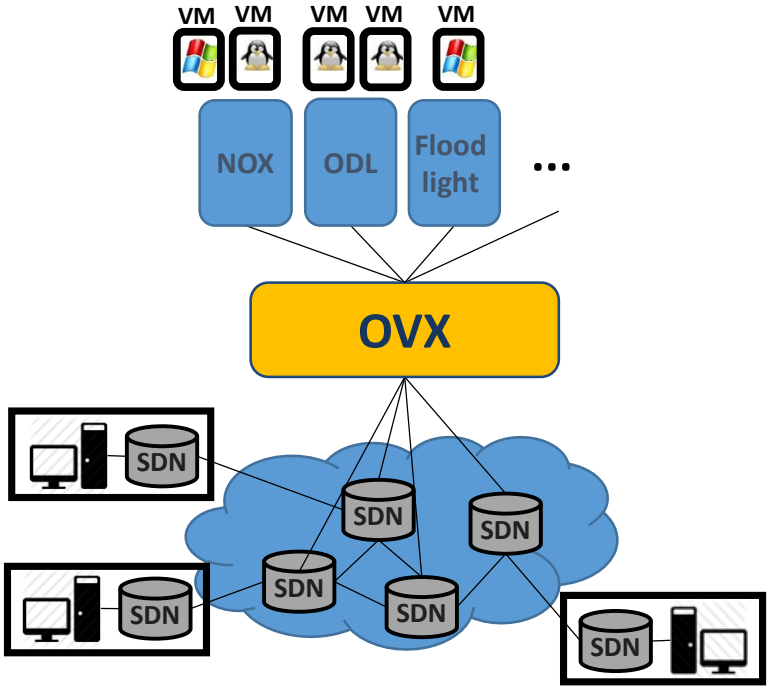


Figure 4.3: OpenVirteX architecture

OpenVirteX thus provides each tenant with the full header space. To achieve this, OpenVirteX places edge switches at the borders of the physical SDN network re-writing the virtually assigned IP and MAC addresses, which are used by the hosts of each tenant, into disjoint addresses to be used within the physical SDN network. With such approach, the entire flowspace can be provided to each virtual network.
FlowN [25] is another proposal that offers full network virtualization, albeit using a different concept. In this platform, tenants can also specify their own address space, arbitrary topology and control logic. Each tenant has full control over its virtual networks and is free to deploy any network abstraction and application on top of the controller platform. However, whereas FlowVisor can be compared to traditional virtualization technology, FlowN is analogous to container-based virtualization, i.e., it is

a lightweight virtualization approach (Figure 4.4). It is designed to be scalable by allowing a unique shared controller platform to be used for managing multiple domains in a cloud environment.

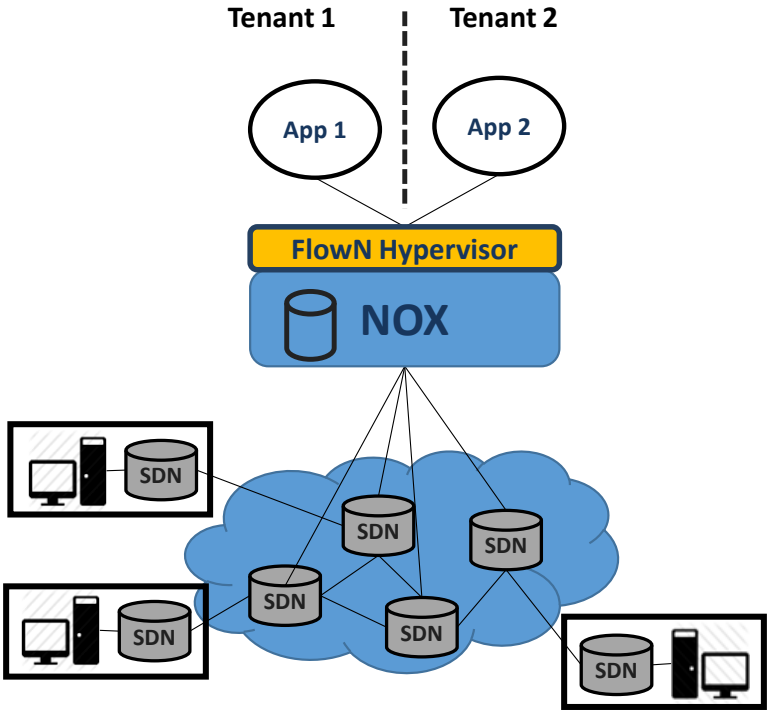


Figure 4.4: FlowN

Similarly to OVX, to achieve address space and bandwidth isolation the solution maps between virtual and physical addresses using the edge switches to encapsulate incoming packets with an extra header (in their implementation VLAN tags were used) to each tenant.
Moreover, FlowN uses database technology to increase the scalability of the mapping between physical and virtual network topologies. The database is used to store the mapping between the physical and virtual resources.
A related effort is the compositional SDN hypervisor [41], a solution designed with a different set of goals from the examples above. Its main objective is to allow the cooperative (sequential or parallel) execution of applications developed with different programming languages or conceived for diverse control platforms. It thus offers interoperability and portability in addition to the typical functions of network hypervisors.

4.5 Edge-based full network virtualization

With the market demand for network virtualization and the recent research on SDN showing promise as an enabling technology, different commercial virtualization platforms based on SDN concepts have started to appear. Their target is multi-tenancy in enterprise datacenters, and interestingly they all follow the same edge-based approach.
VMWare has proposed a network virtualization platform (NVP) [44, 6] that provides the necessary abstractions to allow the creation of independent virtual networks for large-scale multi-tenant environments. NVP is a complete network virtualization solution that allows the creation of virtual networks, each with independent service model, topologies, and addressing architectures over the same physical network. Tenants' applications are provided with an API to manage their virtual networks. NVP's network hypervisor translates the tenants' configurations and requirements into low level instruction sets to be installed on the forwarding devices.

For this purpose, the platform uses a cluster of SDN controllers to manipulate the forwarding tables of the Open vSwitches in the host's hypervisor. Forwarding decisions are therefore made exclusively on the network edge (Figure 4.5). NVP wraps the ONIX controller platform [45], and thus inherits its distributed controller architecture, allowing it to scale. In order to virtualize the physical network, NVP creates logical datapaths, i.e., overlay tunnels, between the source and destination OvSs. After a forwarding decision is made, the packets are tunneled over the physical network to the receiving host hypervisor (the physical network sees nothing but ordinary IP packets).

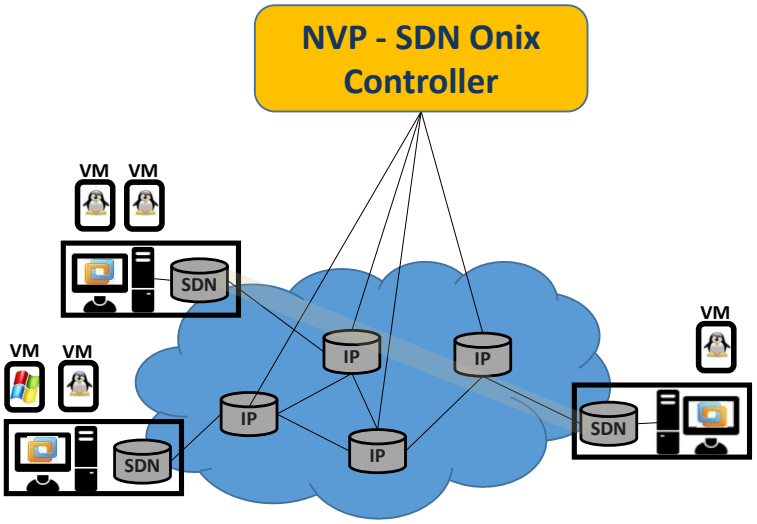


Figure 4.5: Network Virtualization Platform architecture

IBM has also recently proposed SDN VE [62, 49], another commercial and enterprise-class network virtualization platform. SDN VE uses OpenDaylight as one of the building blocks. This solution also offers a complete implementation framework for network virtualization. Like NVP, it uses an edge-based overlay approach, achieving advanced network abstraction that enables application-level network services in large-scale multi-tenant environments.

4.6 Language-based virtualization

Two essential characteristics of virtualization solutions are the capability of expressing modularity and of allowing different levels of abstractions while still guaranteeing desired properties such as protection and isolation. For instance, virtualization techniques can support different views of a single physical infrastructure. As an example, one virtual "big switch" could represent a combination of several underlying forwarding devices. This intrinsically simplifies the task of application developers as they do not need to think about the sequence of switches where forwarding rules have to be installed, but rather see the network as a simple "big switch". This kind of abstraction significantly simplifies the development and deployment of complex network applications, such as advanced security related services.

Pyretic [64] is an interesting example of a programming language that offers this type of high-level abstraction of network topology. It incorporates this concept of abstraction by introducing network objects. These objects consist of an abstract network topology and the sets of policies applied to it. Network objects simultaneously hide information and offer the required services.

Another form of language-based virtualization is static slicing. This is a scheme where the network is sliced by a compiler, based on application layer definitions. The output of the compiler is a monolithic control program that already has slicing definitions and configuration commands for the network. In such a case, there is no need for a hypervisor to dynamically manage the network slices. Static slicing can be valuable for deployments with specific requirements, in particular those where higher

performance and simple isolation guarantees are preferable to dynamic slicing.
One example of a static slicing approach is splendid isolation [35]. In this solution the network slices are made of 3 components: (a) *topology*, consisting of switches, ports, and links; (b) *mapping* of slice-level switches, ports and links on the network infrastructure; (c) *predicates on packets*, where each port of the slice's edge switches has an associated predicate. The topology is a simple graph of the sliced nodes, ports and links. Mapping will translate the abstract topology elements into the corresponding physical ones. The predicates are used to indicate whether a packet is permitted or not to enter a specific slice. Different applications can be associated to each slice. The compiler takes the combination of slices (topology, mapping, and predicates) and respective programs to generate a global configuration for the entire network. It also ensures that properties such as isolation are enforced among slices, i.e., no packets of a slice A can traverse to a slice B unless explicitly allowed.
Other solutions, such as libNetVirt [71], try to integrate heterogeneous technologies for creating static network slices. libNetVirt is a library designed to provide a flexible way to create and manage virtual networks in different computing environments. Its main idea is similar to the OpenStack Quantum project [61]. While Quantum is designed for OpenStack (cloud environments), libNetVirt is a more general purpose library which can be used in different environments. Additionally, it goes one step beyond OpenStack Quantum by enabling QoS capabilities in virtual networks [71]. The libNetVirt library has two layers: (1) a generic network interface; and (2) technology-specific device drivers (e.g., VPN, MPLS, OpenFlow). On top of the layers are the management applications and virtual network descriptions. The OpenFlow driver uses the NOX controller [34] to manage the underlying infrastructure, using OpenFlow rule-based flow tables to create isolated virtual networks. By supporting different technologies, it can be used as a bridging component in heterogeneous networks.

Table 4.1: Virtualization solutions

Solution	**Multiple controllers**	**Slicing**	**QoS "guarantees"**	**Multi-technology**
AutoVFlow [74]	yes, one per tenant	flow space virtualization	no	no, OF only
AutoSlice [16]	yes, one per slice	VLAN tags	no	no, OF only
Compositional Hypervisor [41]	—	—	no	no, OF only
FlowVisor [68]	yes, one per slice	virtual flow tables per slice	yes (VLAN PCP bits)	no, OF only
FlowN [25]	no (contained applications)	VLAN tags	no	no, OF only
IBM SDN VE [62]	yes, a cluster of controllers	logical datapaths	yes (priority-based)	yes (VXLAN, OVS, OF)
libNetVirt [71]	no, one single controller	VLAN tags	no	yes (e.g., VPN, MPLS, OF)
NVP's Hypervisor [44]	yes, a cluster of controller	logical datapaths	yes	no, OVS only
OpenVirteX [9]	yes, one per slice	virtual flow tables per slice	*unknown*	no, OF only
Pyretic [64]	no, one single controller	compiler time OF rules	no	no, OF only
Splendid Isolation [35]	no, one single controller	compiler time VLANs	no	no, OF only

4.7 Summary of reference architectures

Table 4.1 summarizes the virtualization technologies discussed in this chapter. As can be observed, only libNetVirt supports heterogeneous technologies, not restricting its application to OpenFlow-enabled

networks. FlowVisor, AutoSlice and OpenVirteX allow multiple controllers, one per network slice. FlowN provides a container-based approach where multiple applications from different users can co-exist on a single controller. FlowVisor allows QoS provisioning guarantees by using VLAN PCP bits for priority queues. SDN VE and NVP also provide their own provisioning methods for guaranteeing QoS.
To conclude this chapter in Figure 4.6 we present a summary of the reference architectures surveyed. We show the four proposals we consider representatives of the main categories of network virtualization platforms: slicing, proxy-based, container-like, and edge-based, along different axis.

	FlowVisor	OpenVirteX	FlowN	NVP
Full virtualization	No	Yes	Yes	Yes
Type of virtualization	Proxy-based (slicing only)	Proxy-based	Container-like	Edge-based
Centralised controller	Yes	Yes	Yes	No
Arbitrary topology	No	Yes	Yes	Yes
Arbitrary addressing	No	Yes	Partial	Yes
SDN infrastructure requirements	SDN-based network	SDN-based network	SDN-based network	SDN at the edge, IP in the core
Commercial product	No	No	No	Yes
Avaliable open-source	Yes	Yes	No	No
Secure and dependable	No	No	No	No
Multi-cloud	No	No	No	No

Figure 4.6: Summary of reference network virtualization architectures

To the exception of FlowVisor, all proposals enable full virtualization of control, topology and addressing. However, FlowN addressing virtualization is relatively limited as VLAN tags are used for virtual network identification and are hence not made available to the tenant. NVP is centered around the Onix controller and is hence a distributed solution. Contrary to the other proposals, NVP does not require SDN-based equipment in the infrastructure core. The only requirement is at the edge: the VMs run software switches (OvS) that are SDN-controlled. The traditional (IP-based) networking equipment does not need to be replaced nor upgraded. NVP is a closed-source, commercial solution, whereas the other are research-oriented. The FlowVisor and OVX code is made available open-source. Contrary to SUPERCLOUD, all recently proposed work on this area targets a single cloud scenario. In addition, the aforementioned works either do not consider security and dependability in their design, or they do so in a very limited manner. For instance, OVX includes some level of resilience, with a virtual link or switch being mapped onto multiple physical components to provide redundancy. It is in improving the resilience of a network virtualization platform for multi-cloud scenarios that lies the main novelty of SUPERCLOUD.

Chapter 5 A short survey of security incidents and abuses involving public cloud services

In chapter 4, we surveyed existing state of the art network virtualization technologies. We discussed their limitations and their ability to address the requirements of the SUPERCLOUD network virtualization platform that are presented in chapter 2. Before we move forward in this document towards providing a first draft of the network virtualization architecture, we take a step back in this chapter and try to qualify the nature and the main trends regarding the security incidents and abuses that may affect the users of the SUPERCLOUD. Our objective is to shed some light on malicious activities that target current cloud providers, and try to extrapolate in order to identify the main security challenges that need to be addressed by the SUPERCLOUD autonomic security management framework.
While it is widely believed in the security research community that security incidents involving public cloud services are constantly increasing, we are yet unaware of any existing study that measures *the extent at which public cloud services are being affected by those incidents and abuses.* Since the SUPERCLOUD will leverage on ressources from existing cloud providers, there is no reason to hope that things would be any better for SUPERCLOUD users without really understanding and addressing these challenges as part of the autonomic security management framework. Moreover, in the context of SUPERCLOUD, this problem may be amplified as SUPERCLOUD providers would federate heterogeneous resources across multiple physical cloud providers. Each cloud provider would have only a partial view of security incidents that affect its own ressources, and so the tenants in the SUPERCLOUD would need to correlate observations and coordinate actions across multiple providers in order to better characterize and mitigate ongoing attacks and abuses. Therefore, we survey in this chapter the most recent security incidents and abuses affecting public cloud providers, and we further describe in chapter 6 our approach to handle these incidents as part of the autonomic network security management in the SUPERCLOUD.

5.1 Cloud-based security incidents and abuses

A key factor that led to the wide adoption of public cloud services is their flexibility and their straightforward *pay-as-you-go* pricing model where users dynamically create virtual machines at will, provide them with public IP addresses and on-demand compute and storage resources, and then remove them without any sustainable cost. Unfortunately, the downside for this model is the fact of also attracting cyber-criminals, which has paved the way to an active undergroung economy. According to the cloud security alliance, the abuse of public cloud services appears among the top nine most critical threats to cloud computing [51]. In fact, public IaaS clouds provide cyber attackers with virtually unlimited network, compute, and storage resources. Besides, most public cloud providers implement weak registration processes that facilitate anonymity, and so anyone with a valid credit card can easily register and use public cloud services for malicious purposes. For example, an early case of cloud service abuse was publicly uncovered in 2009, where a Zeus command and control (C&C) server was found to be hosted on Amazon EC2 [27]. More recent examples include the SpyEye banking trojan that was found to be using Amazon S3 storage [2], Android malware that exploited the Google Cloud Message service [72], and more advanced persistent attacks that used cloud-based services such as Dropbox and Wordpress as a cover [38]. These incidents constitute a threat against (1) cloud users in

case where users orchestrate attacks to break some security assumptions such as isolation or ressource management [14, 15], and (2) third parties in case where users abuse cloud ressources in order to orchestrate public attacks such as phishing and DDoS.
We conduct in this chapter a longitudinal study over the last six years in order to leverage the nature and main trends of cloud-based security incidents and abuses [36]. This study enables us first to reach out a better understanding of the main security incidents and abuse cases that involve public cloud infrastructures, and then to derive the main requirements for autonomic security management in the SUPERCLOUD. To achieve these objectives, we study several characteristics of the traffic observed between malicious samples in the wild, and some major public cloud services such as Amazon EC2 and Microsoft Azure. Based on our measurements, we discuss the evolution of this phenomenon over the past six years, and we present our key observations and insights into this growing problem.

5.2 Research Approach and Methodology

The key challenge for our survey is about how to qualify security incidents involving public cloud services, and how to obtain a representative record of such incidents that would allow us to draw relevant insights. One posibility would be to monitor ISP network traffic and extract traffic originating from, or towards, cloud-based public IP addresses; and further to isolate and analyze malicious communications. This approach turns out to be very complicated due to the difficulty to obtain a valid ground truth for isolating malicious communications from within the mass amount of benign cloud activity. Therefore, and since malware constitutes by far the mainstream arsenal for cyber attackers and their privileged attack vector [70], we adopt an alternative approach where we directly collect malware communications by analyzing the network traffic recorded by a dynamic malware analysis system. We used for this purpose the Anubis malware analysis system [11], which is a publicly accessible service that analyzes malware samples in an instrumented sandbox. As part of its analysis, the system also records the destinations contacted by each malware sample, and part of the data that is transferred through the connection. Unfortunately, malware can communicate with the cloud for multiple reasons, including malicious activities but also other innocuous connections which range from simple connectivity checks, to the use of public benign services. This greatly complicated the analysis, and required the development of several heuristics to discard malware samples using public services hosted on the cloud, as this cannot be considered an abuse of the cloud itself.

5.2.1 Environmental setup and dataset description

In our experiments, we analyzed the network communication of over 30 million samples, submitted to the Anubis system between years 2008 and 2014. We identified 1.08 million (roughly 3.6%) that connected to at least one publicly routable cloud IP address (namely Amazon EC2 and Microsoft Azure). These IPs were associated to 12,522 distinct cloud-based domains. To identify malicious cloud-based services, we first collected the range of IP addresses assigned to the EC2 and Azure cloud images. Then we tracked all domain names associated with these IP addresses that were contacted at least once by a malicious sample, and we extracted and analyzed their DNS features and the content of network communications between the malware and these domains.
A major challenge in our study is that domain names extracted from the Anubis database do not only include dedicated malicious servers, and so we cannot simply mark as suspicious every connection toward a cloud-based IP address. In fact malware often contacts other public and benign cloud-based services, such as IP lookup services, advertisement websites, and URL shortening. These services are not part of the malicious activity and need to be identified and discarded from our subsequent analysis. On the other hand, another challenge comes from the fact that real malicious domains may have been sinkholed by security organizations at the time the malware was analyzed in Anubis. Malware will be thus redirected towards cloud-based sinkhole services, even though the original domains may have not been hosted on the cloud. Hence, we need to filter these cases and not to consider them as cloud-related malicious activities. Finally, in our experiments we discovered that many malware samples

PPI Domain name	Samples
getapplicationmy.info	116306
sslsecure1.com	71965
oi-imp1.com	68255
secdls.com	52857
oi-config1.com	43526
ppdserver.com	39434
optimum-installer.com	38777
optimuminstaller.com	35510
leadboltapps.net	31918
xtrdlapi.com	18615

PPI Domain name	Samples
torntv.net	16578
powerpackdl.com	15586
oi-config3.com	15578
webfilescdn.com	14050
torntvz.com	12440
premiuminstaller.com	11879
ppdistro.us	10463
bestringtonesmaker.com	10136
baixakialtcdn2.com	9946
oi-config2.com	9601

Table 5.1: Top 20 PPI services in our dataset

were adwares that leverage pay-per-install (PPI) services hosted on Amazon or other cloud providers. PPI services allow software publishers to pay affiliates by the number of installations that they perform on target machines. Although the use of PPI services to distribute malware still constitutes a malicious activity, PPI services are not malicious *per-se*, and so we discard them from our dataset as we only focus on malicious cloud-based services. We address these challenges through a two-steps process, including malware extraction and domain filtering, and that we describe as follows.

5.2.1.1 Malware extraction

We first noticed that a large number of samples in our dataset were executables that leverage PPI services hosted on the cloud. PPI services have recently emerged as a key component of modern cybercrime as they enable to outsource the global distribution of malicious samples. Miscreants supply PPI services with malware executables, which in turn charge them for successful installations based on the requested features for the desired victims.
To identify PPI downloaders in our dataset, we refer to multiple public sources such as PPI forums [1] and public PPI web sites. The main challenge was to identify the different PPI brands, since there are new PPI brands that constantly appear over time. To address this challenge, we analyzed the public PPI services that were mostly contacted by the samples in our dataset, and we tried to infiltrate these services by supplying a small program we developed for distribution. By testing and manually reverse engineering the resulting installer we developed a set of 13 distinct network signatures that match the download URLs associated with different families of PPI services. By using these signatures on the malware traffic we could further discard their associated samples in our dataset. Overall, we discarded up to 90% of our initial dataset. Table 5.1 summarizes the top 20 PPI domain names that were contacted by malware in our dataset and the number of samples associated with each service.
In addition to PPI downloaders, our dataset also includes benign files that were submitted for analysis in Anubis. In fact Anubis is a public service where Internet users freely submit suspect files for analysis. These files may turn out to be benign files that connect to benign cloud-based services and so they also need to be discarded from our dataset. Since our dataset covers a period where the most recent samples are few months old, we use anti-virus (AV) signatures to identify and discard benign samples. We refer to public services such as VirusTotal[1] to scan our dataset, and we consider as benign files all samples that are detected by less than five AV editors. Our dataset finally includes $45,422$ confirmed malicious malware samples, which corresponds to almost 5% of our initial dataset. The remaining samples were discarded as we do not have enough confidence about their malicious nature.

5.2.1.2 Domain filtering

Domain filtering aims at discarding from our dataset all domains that are associated with benign cloud-based services. Although these domains supply public Internet services that can be used by

[1] `http://www.virustotal.com`

Service		Domain Names	Malware Samples
Public Services	Advertising	930	22,216
	File sharing	796	7,657
	Domain redirection	270	479
	Others	211	1,723
Sinkholed		26	4,249
Infected		22	231
Dedicated		1,648	7,884
		N/A	983
Total		**3,903**	**45,422**

Table 5.2: Cloud-based service categories

malware, they are not part of dedicated malicious services. Out of the initial set of 12,522 distinct cloud-based destinations, malware extraction discarded 8,619 benign cloud-based services. We classify the remaining 3,903 domains into four categories, as illustrated in Table 5.2.
The first category includes public benign services that were contacted by malware. We found multiple examples in this category, such as public IP resolvers (e.g. `hostip.info`), advertising and affiliate services, file sharing (e.g. `dropbox`) and URL shortening (e.g. `notlong.com`). To identify known public services in our dataset, we leverage multiple sources such as the Alexa list of top domains, public repositories that provide URL shortening services (e.g. `bit.do`) and file sharing. We also refer to AV labels in VirusTotal in order to identify generic adwares, and we identify as advertisement services all domains that were only contacted by Adwares samples.
The second category includes domain names that have been sinkholed, and so they were redirected to sinkhole destinations that are hosted on the cloud. EC2 hosts multiple sinkhole destinations that are used to subvert BOT communications with their remote C&C domains. These domains were not originally hosted on the cloud, and so they need to be discarded from our dataset. We leverage the `X-sinkole` HTTP header[2] in order to identify sinkhole destinations.
The last two categories include both dedicated malware domains and domains that were once infected and temporarily used as part of the malicious infrastructure. The separation between these two categories is more difficult and more prone to errors. We rely on multiple empirical observations in order to discriminate between the two cases. First, we assume that dedicated malicious services that were hosted on the cloud more than one year ago have all been detected or abandoned at the time we run our experiments. Based on this assumption, we actively probe all the domains and if the domain is still in use and points to a populated web page, we classify it as an infected host. On top of this first heuristic, we also leveraged the history of DNS requests towards expired domains in order to assess the average lifetime of these domains. We use for this purpose DNS records that we extracted from DNSDB, a passive DNS duplication service[3]. Our assumption is that infected domains are expected to appear in DNS records a long time before the associated malware first appears in the wild. Dedicated domains instead, usually appear a short time before the malware is released and go offline a short time after the malware has been detected. By combining these two heuristics we were able to identify 22 infected cloud-based services over the six years of observation. We identified the remaining 1,648 domains as being associated with dedicated malicious services. Last of all, most of the connections were initiated using a domain name, but 983 malware samples directly connected to cloud-based IP addresses that were hard-coded in the malware itself, without any prior DNS request.

5.3 Main Observations and Insights

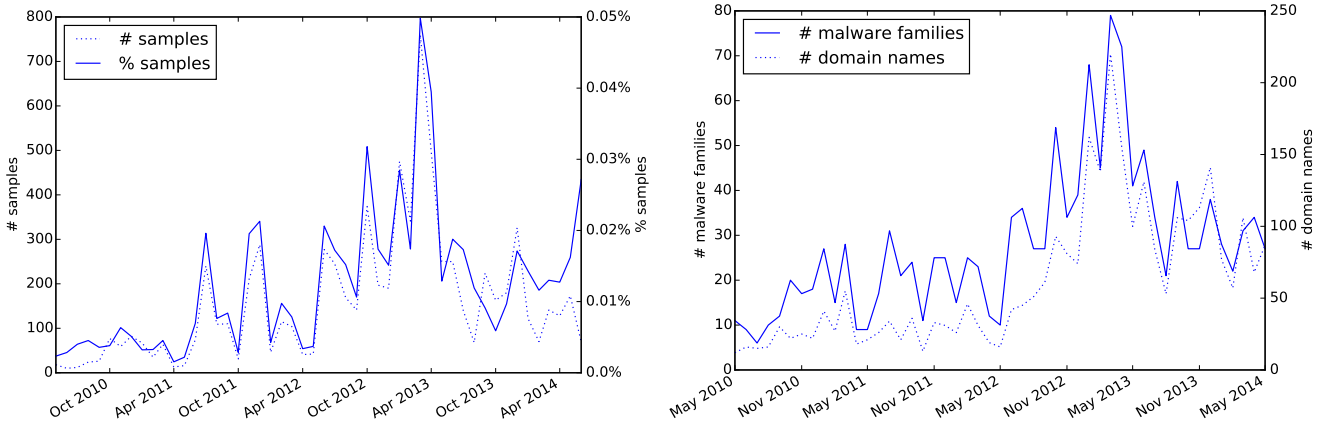
Our resulting dataset in table 5.2 allows us to analyze both the malware families that are using public cloud services in some capacity, and the distribution and lifetime of malicious services in the cloud. A

[2] `http://www.iss.net/security_center/reference/vuln/HTTP_Malware_XSinkhole.htm`
[3] `https://www.dnsdb.info/`

AV label	# samples
Downloader Fosniw	1249
Worm Vobfus	909
Android DroidAp/SmsSend	634
Downloader Murlo/Renos	567
Backdoor QQRob	528
Downloader Small BKY	208
Delf Downloader	196
Trojan Injector	194
Downloader 8CCBF09D99CF	186
Clicker Agent	172

AV label	# samples
Trojan Kryptik	160
Ramnit	129
Downloader Banload/Zlob	128
Trojan Kazy	127
Downloader Virut/Virtob	127
Zbot	117
Malware SoftPulse	108
Downloader Karagany	90
Trojan Krap	89
Downloader Cutwail	80

Table 5.3: Top 20 malware family



(a) Number and ratio of distinct malware samples that connected to dedicated cloud-based malicious services

(b) Number of distinct malware families that used dedicated cloud-based malicious services and number of their associated malicious domains

Figure 5.1: Malware dataset analysis

first question that we would like to address is whether the use of public cloud services is still limited to a small set of malware attacks, or whether it can be generalized to different families of malware. To answer this question, we analyze the 8,867 distinct malware samples that we found to be connecting to dedicated malicious machines on the cloud. Since our dataset includes malware samples that are at least few months old at the time we run our analysis, we believe it is reasonable to use AV labels as a reference to understand our dataset. More complex behavioral clustering mechanisms, as proposed for instance in [10] and [56], could be applied to refine the classification. However, since we only seek a broad understanding of the major attacks that use cloud services and we can tolerate few misclassification errors, we opt for a simpler AV-based solution.
Using our approach, we were able to identify 377 distinct malware attacks. As clearly illustrated in Table 5.3, which provides the list of top 20 malware families, we were not able to identify a predominant malware family that uses dedicated malicious cloud services. More interestingly, our dataset includes malware that uses different topologies, including also decentralized peer-to-peer networks such as the Sality malware. Clearly the use of dedicated malicious cloud services is not limited to a small set of malware families, but it could be generalized to all categories of malware.

- **First requirement in the context of SUPERCLOUD:** Since the security incidents that may affect public cloud services belong to very different categories of attacks, the users of the SUPERCLOUD should be able to compose the security services that are most appropriate according to the context and security requirements of their use cases that are hosted on the SUPERCLOUD.

On the other hand, and since the hosting and usage of malicious services on public cloud infrastructures

is not limited to specific malware families, our next goal is to identify if there is a clear trend on the amount of malicious software that make use of cloud services. Figure 5.1a illustrates the number of distinct samples that connected to dedicated malicious domains hosted on the cloud during the period of our observation. To account for changes in the overall number of submissions, the figure also shows the percentage of distinct samples compared to the total number of samples submitted to Anubis in the same period. Figure 5.1b shows the number of distinct malware families and the number of their associated malicious domains that were hosted on a public cloud over the same period.
On average, the number of malware that uses dedicated cloud-based malicious services has grown by almost 4 times between 2010 and 2013. The overall trend also includes multiple peaks, that after a manual analysis resulted to be associated with multiple instances of malicious servers found to be temporarily hosted on Amazon EC2. While the fast growing number of malware samples that use cloud-based services may appear as a natural consequence of the general increase in the number of malware attacks [4], Figure 5.1a shows that this is not the case and that the ratio between these malware samples and the total number of malware submitted to Anubis has been increasing at the same rate. As illustrated in Figure 5.1b, this trend can be generalized to all malware families, which means *there is a growing appetite towards using cloud infrastructures to host malicious servers*. This could be due to multiple elements, including the fact that cloud services have been rapidly expanding in the past few years, and the fact that they are easy to access and still lack a strict accountability and control over their hosted machines [42].

5.3.1 Role of Public Cloud Services in Malware Infrastructures

In this section we describe the different ways malicious software makes use of public cloud services. In particular, we are interested in understanding whether miscreants specifically target cloud services or whether they use these services as small parts in a much larger redundant infrastructure.
For this purpose, we measured the ratio of remote malicious destinations that were hosted on the cloud, compared to all malicious destinations contacted by the malware during the analysis. Then, for those malicious services that were hosted on the cloud, we determined if they were hosted on the cloud only as part of a redundant mechanism. In this case, we extracted the DNS requests executed by the malware and we monitored the DNS records history using the DNSDB service in order to compute the ratio of IP addresses that belong to the cloud IP range, compared to all IP addresses associated with that malicious domain in other moment in time. This technique works particularly well in the presence of round-robin DNS and DNS fast-flux techniques that are often adopted by attackers. For instance, miscreants can associate different IP addresses with the same malicious domain name, where only some of these IPs may be hosted on the cloud.
Figure 5.2 presents the average distribution of the ratio of remote malicious destinations that were hosted on the cloud, compared to all malicious destinations contacted by all malware samples. We present our findings as a box plot where malware samples are classified according to their submission date to Anubis. The Y-axis characterizes the ratio of dedicated malicious domains that were hosted on the cloud, compared to all malicious domains contacted by malware. Since we included in this experiment only malware samples that used dedicated malicious services on the Cloud, the percentage is always greater than 0%. On the other hand, a malware would fit into the 100% category in case all dedicated malicious domains that were contacted by the malware were hosted in the cloud.
As shown in Figure 5.2, miscreants mostly use public cloud services in order to host only certain components of their malware infrastructures. Note that while in 2010, and for malware that uses EC2 to host its dedicated malicious services, only few components of its malware infrastructures were found to be hosted on EC2 (less than 40% of remote malicious domains in average); *the use of public clouds to host dedicated malicious services has rapidly evolved in the recent years, including malware samples that were found to be exclusively communicating with dedicated cloud-based malicious domains in years 2013 and 2014*. In other terms, miscreants have been recently referring to public cloud services in order to setup and manage their entire malware infrastructure. Therefore, although the use of public cloud services is still limited to only specific components of malware infrastructures, we observe an

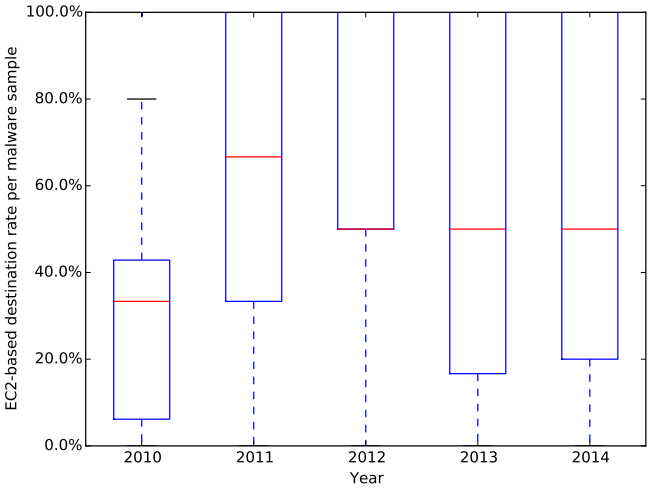
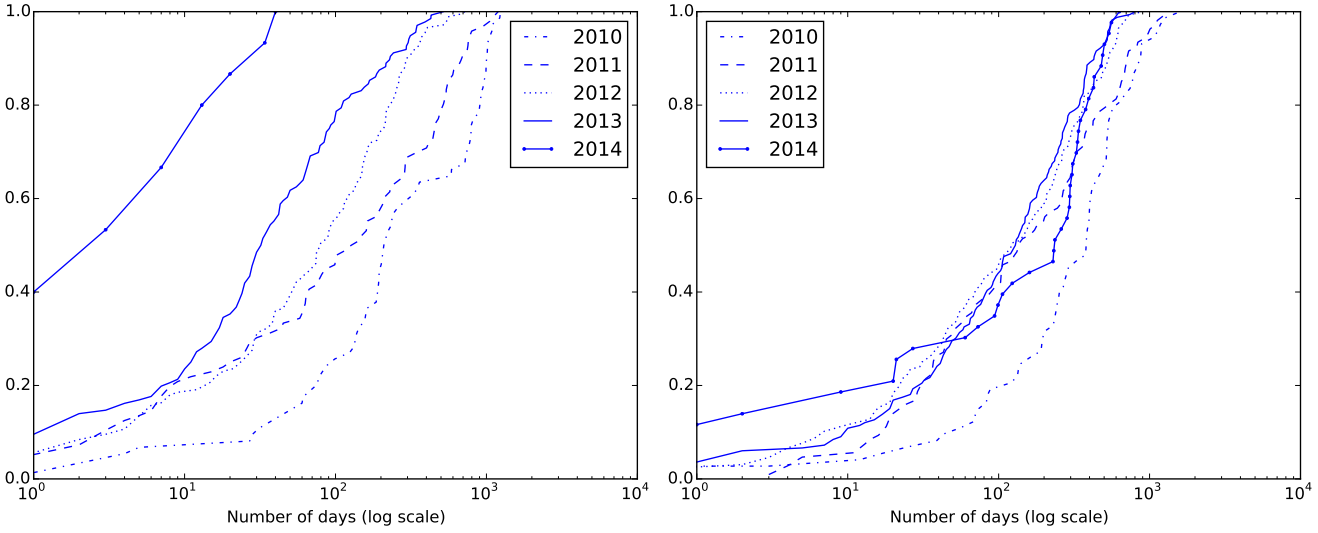


Figure 5.2: Rate of dedicated malicious cloud-based domains contact per malware sample



(a) Time elapsed until a dedicated malicious EC2-based domain was first observed in Anubis

(b) Time until a dedicated malicious domain was no longer hosted on EC2, after it was observed in Anubis

Figure 5.3: Lifetime of dedicated malicious EC2-based domains

increasing appetite for miscreants towards using public cloud services to setup and manage additional components of their malware infrastructures.

- **Second requirement in the context of SUPERCLOUD:** Since miscreants are moving additional components of their malicious infrastructures in the cloud, and when these components may end up on different cloud providers, users of the SUPERCLOUD should be able to correlate features and alerts across multiple cloud providers in order to better characterize ongoing attacks and abuses.

5.3.2 Dedicated Domains Lifetime Estimation

In the last part of our study, we tried to estimate the average time that malicious domains persist on the cloud. Our approach leverages the lifetime of the cloud-based malicious domains in order to estimate whether the use of public cloud providers adds more resilience to malware infrastructures.

In the following, we refer to the lifetime of a cloud-based malicious domain as the duration when it was consecutively associated with a cloud IP address in the passive DNS records. Note that the use of passive DNS service only provides an estimation of the real lifetime of these domains, but this is an approximation that is often used for this type of measurements [50]. Since domains first appear in the passive DNS services when they are actively requested on the Internet, we consider that the use of this service provides a reliable estimation of the real duration in which a given domain remained active and accessible on the wild. In this section, we observe only dedicated malicious domains that were hosted in the cloud, and that were contacted by our malware dataset collected over a period that ends by June 2014. Hence, we only consider historical DNS records associated with malicious servers that are no longer hosted on the cloud at the time of writing. Note that these domains may be still accessible but no longer associated with any public cloud-based IP address.
We defined two metrics for our experiments. First of all, we measured the time between the domain first appeared in the passive DNS service and the time the malware was analyzed by Anubis. Second, we extract the time when a dedicated malicious domain is no longer associated with an EC2 IP address, after the malware was first submitted to the Anubis service.
The results of our experiment are summarized by the cumulative distributions that are illustrated in Figure 5.3. The graphs separately illustrate the results of our experiments for the last five years since 2010, in order to extrapolate some trends and assess the efficiency of security measures implemented by public clouds over time. The first graph shows that the distribution is clearly moving toward the top-left corner, meaning that each domain was observed in Anubis soon after it first appeared in the wild. For instance, while in 2011 around 50% of the domains were already present in the passive DNS service (and therefore active in the wild) for 100 days before some malware in Anubis contacted them, in 2014 they had only been active for two days. In other words, the security community became very efficient to promptly detect, collect, and submit malware samples.
Unfortunately, Figure 5.3b shows that the time these domains were hosted on the cloud *after* the malware was analyzed remained stable over the same period. While many factors are involved in this process, this seems to suggest that Cloud providers did not improve their ability to detect and report abusive behaviors. In other words, *our observations suggest that the security mechanisms implemented by public cloud service providers have not contributed to reducing the lifetime of malicious domains hosted by these providers.*
To confirm these findings, and since cloud providers may take-down malicious IPs and not their associated domain names, we analyzed the way malicious EC2 domains resolve to different IP addresses over time. We evaluated how long malicious machines remain active on EC2 before they are taken down by the cloud provider, and so miscreants may be forced into migrating their malicious domains towards other IP addresses. We monitored for this purpose all DNS records in the DNSDB service, searching for different IP addresses that were associated with every malicious domain in our dataset. Confirming our hypothesis, we found multiple instances of malicious machines that remained active on EC2 even for several months before they were migrated towards other IP addresses in the cloud.
As illustrated by the examples in Table 5.4, the first two malicious domains were associated with the same EC2 IP address for up to twenty consecutive months before they went out from the EC2 IP ranges. Interestingly, certain malicious domains, such as domains 1 and 2, as well as domains 3 and 4 in Table 5.4, were associated with the same IP address during the same period of time, which seems to indicate that miscreants may associate different domain names with their malicious machines in the cloud in order to obtain a better resilience against domain blacklisting. Moreover, *they also seem to benefit from the flexibility offered by the cloud in order to migrate towards new IP addresses in EC2 as soon as their current IP addresses have disappeared from the active DNS records, which may suggest that their malicious machines have been identified and taken down by the cloud provider (EC2 in our study).* In total, we observed similar behaviors in over 240 malicious domains in our dataset.

- **Third requirement in the context of SUPERCLOUD:** Since malicious servers tend to frequently migrate within the same address space in the cloud, users of the SUPERCLOUD should be provided with appropriate monitoring applications that enable them to monitor a

Id	Domain	IP address	First seen	Last seen	Duration (months)
1	09sp.co.tv	174.129.222.176 174.129.242.247	August 2010 January 2011	October 2010 November 2012	3 22
2	47gr.co.tv	174.129.222.176 174.129.242.247	July 2010 February 2011	July 2010 November 2012	1 21
3	dl.ka3ek.com	107.20.206.69 54.209.129.218	January 2013 January 2014	November 2013 January 2014	11 1
4	hightool.com	107.20.206.69 54.209.168.250 54.208.247.222	January 2013 March 2014 September 2014	December 2013 September 2014 September 2014	12 7 1
5	hzmksreiuojy.com	54.241.7.53 50.18.179.196 50.17.195.149	April 2013 April 2013 July 2014	April 2013 October 2013 July 2014	1 7 1

Table 5.4: Examples of domains that rotated their IP addresses on EC2 over time

service each time it migrates towards a different VM. Moreover, SUPERCLOUD users should be also provided with appropriate tools that enable them to promptly react upon detection of malicious services.

5.4 Discussion and conclusion

The experiments conducted in this chapter show that the use of public cloud services by malicious software has been increasing over the last six years, despite the measures taken by certain providers to limit the extent of these incidents. They also seem to suggest that the use of dedicated malicious cloud services can be generalized to most attack categories, which led us to derive a first requirement for the network security management in the SUPERCLOUD.
Another observation relates to the fact that miscreants are increasingly moving additional components of their malicious infrastructures into the cloud. Therefore, and when this problem is extrapolated towards SUPERCLOUD infrastructures, SUPERCLOUD users would need to correlate features and alerts across the multiple underlying cloud providers in order to better characterize ongoing attacks and abuses. This constitutes a second requirement for the network security management in the SUPERCLOUD.
The final observation of our study is related to how the cloud providers, Amazon in our case, respond to this threat. Even though we did not have a direct way to observe their reaction, we were able to measure for how long - after the malware was publicly known - the malicious domains it contacted were resolving to IPs hosted on EC2. While the absolute value is not very important, the fact that it remained constant over the past four years seems to indicate that the cloud provider did not make any substantial improvement in detecting and handling security incidents and abuse cases that may occur within the cloud infrastructure. This led us to derive yet a third requirement for the network security management in the SUPERCLOUD.
In the next chapter, we discuss the design requirements and the SUPERCLOUD network architecture components that enable us to address those three requirements.

Chapter 6 Preliminary Network Virtualization Architecture

In this chapter we present a preliminary version of the resilient multi-cloud network virtualization architecture to be developed in SUPERCLOUD. The design of the architecture aims to fulfill all the requirements defined in Chapter 2. The architecture follows an SDN approach (Chapter 3) and is informed by the recently proposed platforms for network virtualization (surveyed in Chapter 4). These platforms share a few characteristics. First, they target cloud environments. Second, the use of SDN and logically centralized control to achieve some form of network virtualization (from slicing to full virtualization of control, address space and topology). The novelty of the SUPERCLOUD solution arises from the use of multiple clouds, its particular focus on resilience (security and dependability), and the integration of mechanisms in its design such as network snapshot and autonomic security management, that have hitherto been provided standalone and that have not yet considered the specificities of multi-cloud environments.

6.1 Design principles

In this section we detail the design principles and techniques proposed to fulfill each of the requirements defined in Chapter 2.

6.1.1 Remote, flexible control of network elements

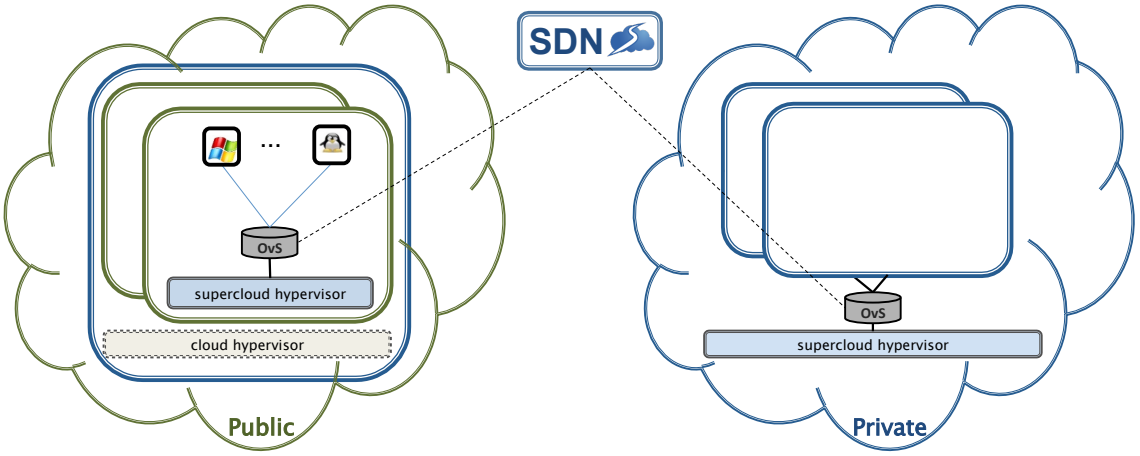


Figure 6.1: SDN logically-centralized control over OvS switches

To fulfill this requirement we intend to base our solution in the Open vSwitch (OvS), a software switch for virtualized environments that resides within the hypervisor or management domain. OvS exports an interface for fine-grained control of packet forwarding (via OpenFlow) and of switch configuration (via OVSDB). This allows SDN-based logically centralized control. Open vSwitch has broad support

from Operating Systems (Linux, FreeBSD, NetBSD, Windows, ESX), virtualization software (KVM, Xen, Docker, VirtualBox, Hyper-V), and cloud software (OpenStack, CloudStack, OpenNebula). OvS is already widely used, being the most popular OpenStack networking backend and the default network stack in XenServer.
Figure 6.1 illustrates the placement of OvS in the architecture. In public clouds, SUPERCLOUD will not have access to the cloud hypervisor, and thus OvS will be part of the SUPERCLOUD hypervisor that runs inside each VM. Private clouds include the supercloud hypervisor running on bare metal and also OpenFlow hardware (not shown in the figure). An SDN controller will establish TCP or TLS connections with each SDN switch (OvS software switches and hardware OpenFlow switches) to control the forwarding plane. The controller server will also run OVSBD to configure the switches (ports, tunnels, etc.).
Design requirement **DR1 Network Controllability** is thus fulfilled with SDN logically-centralized control over the forwarding and configuration state of Open vSwitches.

6.1.2 Security and dependability

A resilient network virtualization platform requires resilience at both the data and the control planes. We consider several techniques for the resilient data plane: multipath routing, network coding, and (secure) path monitoring. There is already literature on traditional techniques for resilient communications [75, 73], multipath routing [8, 54, 29], and on network coding [43] we will leverage on. SDN-based monitoring is a relatively new topic, so novel solutions will be required to make it secure.
For the control plane the controller itself, as the core component of the virtualization platform, needs to be resilient. The SUPERCLOUD controller will be fault-tolerant. We also plan to investigate the possibility of giving strong guarantees of consistency and correctness. The consistency of both data and control planes (and their interaction) will be necessary to avoid network anomalies (loops or/and security breaches).
By increasing the security and resilience of the data plane and the dependability of the control plane we fulfill design requirements **DR2 Network Availability** and **DR3 Network Security**.

6.1.3 Full network virtualization

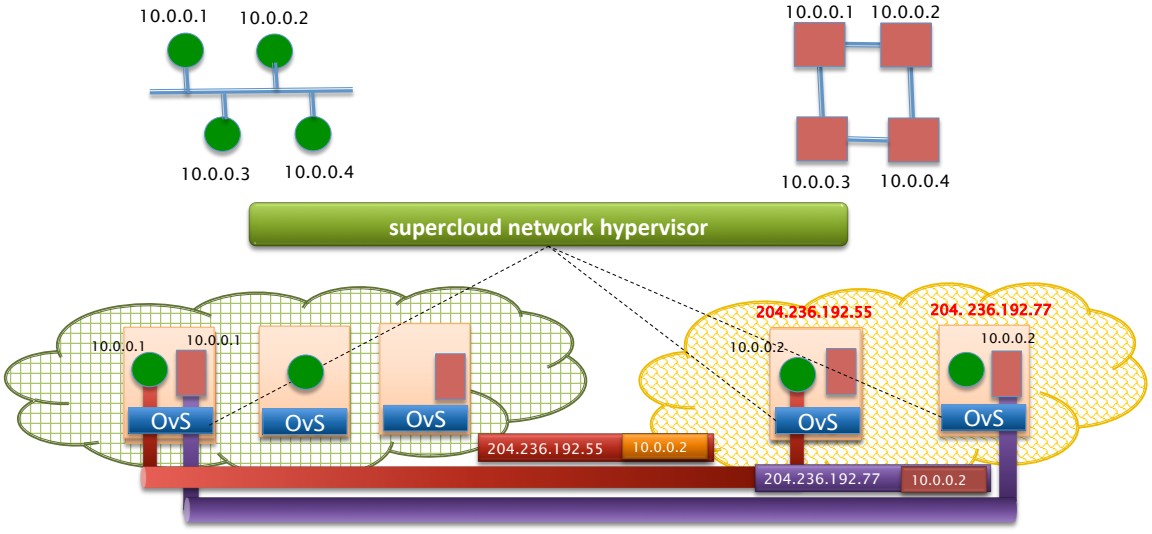


Figure 6.2: An approach to virtualize network addressing

The SUPERCLOUD network virtualization platform has to guarantee isolation between tenants, while enabling them to use their desired addressing schemes and topologies. For isolation, the logical cen-

tralization of control allows flow rule redefinition at the edge of the network and translation/mapping of physical/virtual events/requests in the logically-centralized network hypervisor.
In Figure 6.2 a particular approach of how the addressing abstraction may be implemented is presented. In the figure, two SUPERCLOUD users define different topologies but use the same addressing scheme. For the physical network to be shared all packets that leave a SUPERCLOUD user VM need to either be tagged (e.g., using VLAN tags or MPLS labels), be re-written, or the packets need to be tunneled. Tagging is simpler to implement but reduces the service options for the user (for instance, if VLAN tags are used for virtualization the user cannot employ VLANs in his or her network). Rewriting the IP and/or MAC addresses allows the full header space to be used by the tenants, at the expense of increased complexity in mapping virtual to physical addresses. Creating tunnels between VMs (as shown in the figure) is another alternative that allows the full header space to be used by SUPERCLOUD users. The main drawback is the overhead introduced by the encapsulation process. Either of these techniques is supervised and controlled by the SDN controller. We plan to explore all three alternatives.
For topology abstraction one possibility is to intercept all LLDP (Link Layer Discovery Protocol) messages. LLDP is the protocol used in OpenFlow networks for network discovery. By intercepting all topology-related messages the SDN controller can offer arbitrary virtual topologies to SUPERCLOUD users.
Based on the study of the SDN-based network virtualization platforms presented in Chapter 4 and on the multi-cloud setting, we have considered two architectures that are able to fulfill the full network virtualization requirement: a container-like and a proxy-based approaches. The container-like architecture (Figure 6.3) is a "special" application that runs on the SDN controller to map the physical/virtual events/requests. This application can be made to intercept the flow of messages between the physical network and the users' virtual network, and vice-versa, thus enabling full network virtualization.

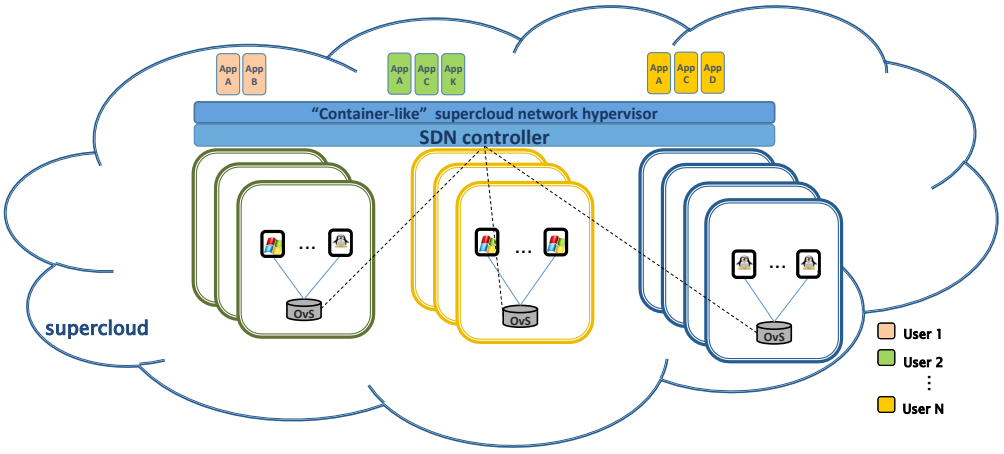


Figure 6.3: Container-like architecture

This architecture enables two service types, presented in Figure 6.4.

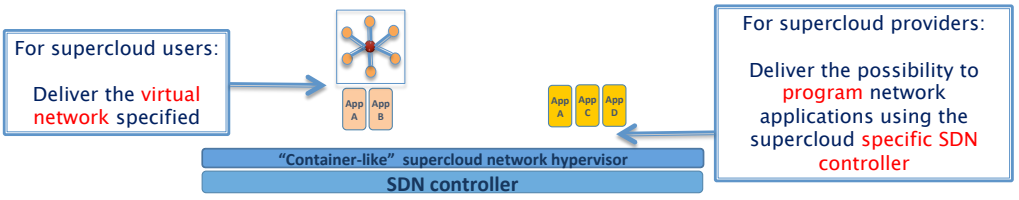


Figure 6.4: Services provided by the container-like approach

For SUPERCLOUD users this architecture is able to deliver the virtual network specified. SUPER-

CLOUD users may not want to program their own network applications, so this seems to be enough for this use case. For SUPERCLOUD providers the architecture delivers the possibility to program network applications using the SUPERCLOUD-specific SDN controller. This allows SUPERCLOUD providers to build their own network applications, but restricts them to use a specific controller.
In the proxy-based architecture (Figure 6.5) the proxy is a "special" SDN controller that maps the virtual/physical events/requests from the controllers (above) and the OpenFlow switches (below). By intercepting all messages it also provides full network virtualization.

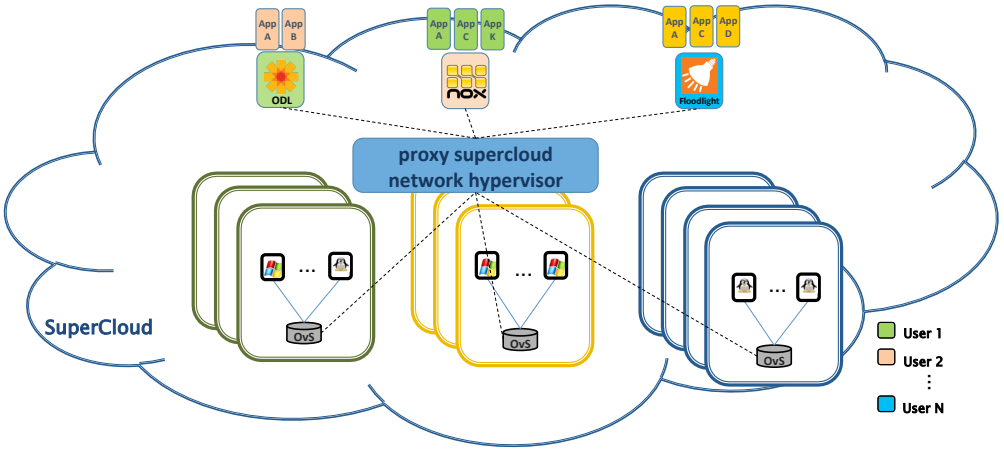


Figure 6.5: Proxy-based architecture

Differently from the container-like solution this architecture enables three service options, presented in Figure 6.6. For SUPERCLOUD users, again, this architecture delivers the virtual network specified. Similarly to the container-like architecture, for SUPERCLOUD providers it delivers the possibility to program network applications using the API of the SUPERCLOUD solution. In addition, it also allows the provider to program the network using the provider's own SDN controller and applications.

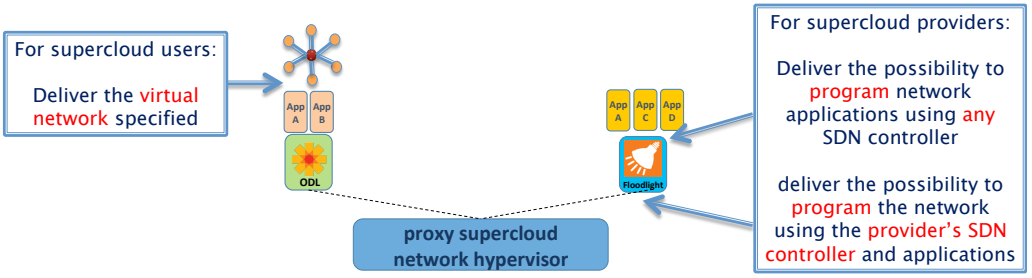


Figure 6.6: Services provided by the proxy-based approach

In summary, the container-like architecture uses a shared controller platform, and thus:

- Is a lightweight solution, as only one SDN controller is needed for all SUPERCLOUD users.
- May be relatively simpler to implement because the application runs on top of an SDN controller. Programming based on the API offered by an SDN controller is much simpler than programming based on "raw" Openflow messages.
- Is simpler to replicate, since the controller and the network virtualization application reside in the same host. This facilitates the design of a fault-tolerant architecture.
- The main limitation is that SUPERCLOUD providers are constrained to build their applications using the API of the SUPERCLOUD SDN controller.

On the other hand, the proxy-based architecture offers:

- A higher level of programmability (for SUPERCLOUD providers), as they can use their own SDN controllers.
- However, it is heavyweight, requiring one SDN controller per user.
- It is also harder to replicate because the SUPERCLOUD network hypervisor (the proxy) is typically external to the different controllers, increasing the complexity of the distributed protocols required for fault tolerance.
- It is relatively harder to implement as it requires low-level OpenFlow programming (although one could leverage on OpenVirteX [9], an open-source solution).

Based on the characteristics of these two reference architectures, we currently tend towards the container-like one as the basis of our solution. The fact that it is a lightweight solution offers better perspectives in terms of scaling. Also, implementing the solution using a controller that offers an expressive API gives more confidence in accomplishing our goals in a reasonable timeframe. Finally, as security and dependability are core requirements of our architecture, a reduction of the complexity of the necessary distributed protocols is an important advantage.
By employing SDN control and the virtualization techniques presented above we attain network isolation, topology and addressing virtualization, and thus fulfill design requirement **DR4 Network Virtualization**.

6.1.4 Autonomic security management

The SUPERCLOUD network virtualization platform provides users with isolated network environments where they can define they own addressing schemes and supply their own set of applications. These applications, as witnessed in chapter 5 of this document, are getting increasingly targeted by attacks and cloud abuses, which may affect both the tenant owner, as well as other tenants in case where attackers manage to break basic security assumptions such as isolation between tenants in the SUPERCLOUD. Therefore, the autonomic security management framework capitalizes on recent advances in intrusion detection capabilities (e.g. [14] and [15]) in order to address the main requirements that we described in chapter 5.
In order to achieve such a user-centric management of network security, the SUPERCLOUD will satisfy two design specifications. First of all, an appropriate SDN security control application will be provided on top of the network abstraction layer. It allows users to compose their own security services in the data plane, and to tune these services on a per-flow (e.g. web traffic) or per-destination (e.g. towards a database service) basis. Second, users will be able to define their own security management procedures that will be automatically triggered by the SUPERCLOUD in response to attacks and security incidents.
In order to meet the first design specification, we will explore available SDN controller APIs that make possible the implementation of an SDN security management application on top of the SUPERCLOUD network abstraction layer. This application provides an abstraction model that enables to represent all security services available in the data plane (e.g. IDS, Proxies, honeypots, anti-DDoS). It further enables users to associate, on-demand, flows in the data plane with their appropriate security services. Lastly, this application will dynamically manage routing policies with respect to the security management procedures and the requirements that are introduced by the SUPERCLOUD users.
Regarding the second design specification, we will explore possible techniques to provide the SDN security management application with the ability to collect and process security incidents (e.g. standard alert formats such as IDMEF), to correlate them with other information related to the network topology, and then to trigger appropriate user-defined reaction strategies.
By meeting these two design specifications we fulfill design requirement **DR5 Autonomic Security Management**.

6.1.5 Network snapshot

VM snapshot is an important management tool used in public and private clouds. We plan to leverage on well-proven techniques for VM snapshotting and extend them for our multi-cloud setting. In addition, we plan to snapshot not only the VM, but also the network state. The motivation is the fact that a VM rarely acts alone, and snapshotting a single VM in isolation would mean losing a significant portion of the complete state that pertains a service.
The first step in snapshotting an SDN switch is the flow table, the list of pattern-action rules. Flow table rules include not only the actions that match a certain packet header pattern, but may also include traffic counters and timers for deleting expired rules. In addition to the flow table rules, the switch configuration and its queues also need to be snapshotted. The latter include the common packet queues found in network switches and the set of packets that are awaiting instructions from the SDN controller.
By leveraging on well-proven VM state snapshot techniques, and by extending them with network state, we fulfill design requirement **DR6 Network Snapshot**.

6.1.6 Scalability

To increase the scalability and performance of the network virtualization platform several techniques will be explored. First, to avoid it being a bottleneck, the SUPERCLOUD controller will be distributed. As distribution brings with it network state consistency challenges, we aim to explore techniques to guarantee consistency and correctness between the different controller instances.
To improve performance we will further explore efficient algorithms for virtual network embedding and techniques to allow fast virtual/physical mapping. Another avenue of work could be to couple storage solutions with networking (e.g., using deduplication techniques in SDN).
By distributing the network control plane and employing efficient algorithms for mapping and embedding we fulfill design requirement **DR7 Network Scalability**.

6.1.7 Compatibility and interoperability

For compatibility and interoperability we plan to investigate OASIS Topology and Orchestration Specification for Cloud Applications (TOSCA). This open standard defines the interoperable description of services and applications hosted on the cloud. It allows expressing application requirements independently from cloud provider capabilities, and hence enables flexible movement between different clouds.
We plan to fulfill design requirement **DR8 Network Interoperability** by leveraging on OASIS TOSCA or related open standards for cloud interoperability.

6.2 Preliminary architecture

The preliminary SUPERCLOUD network virtualization platform architecture is shown in Figure 6.7. The SUPERCLOUD network hypervisor controls and configures the OvS switches that are installed in all VMs (along with the OpenFlow hardware switches, not shown in the figure). This hypervisor is built as an application that runs in the SUPERCLOUD SDN controller, therefore following a container-like approach. Each cloud will have a specific VM that will establish secure tunnels with all other clouds. We call this VM the proxy (despite the same name, this is not to be confused with the core component of the proxy-based architecture presented in Section 6.1.3). In a distributed configuration the proxy will possibly host an instance of the SDN controller. For each tenant a specific set of network applications that control the tenants virtual network will run on top the SUPERCLOUD network hypervisor, which is responsible for mapping the virtual and physical resources.
The design of the hypervisor software follows a modular approach. We present the initial building blocks in Figure 6.8. The platform is divided into three main blocks: the graphical user interface

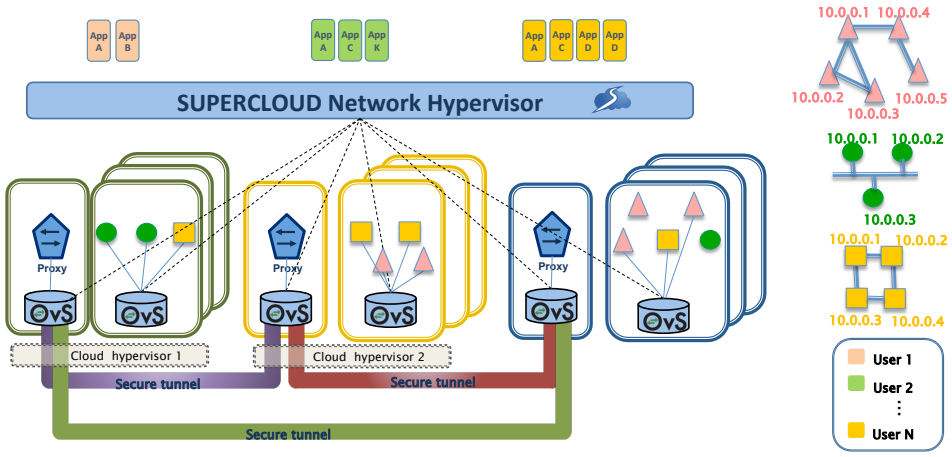


Figure 6.7: Preliminary network virtualization platform architecture

(GUI, on top), the configuration block (left side, in green), and the runtime block (right side, in blue).

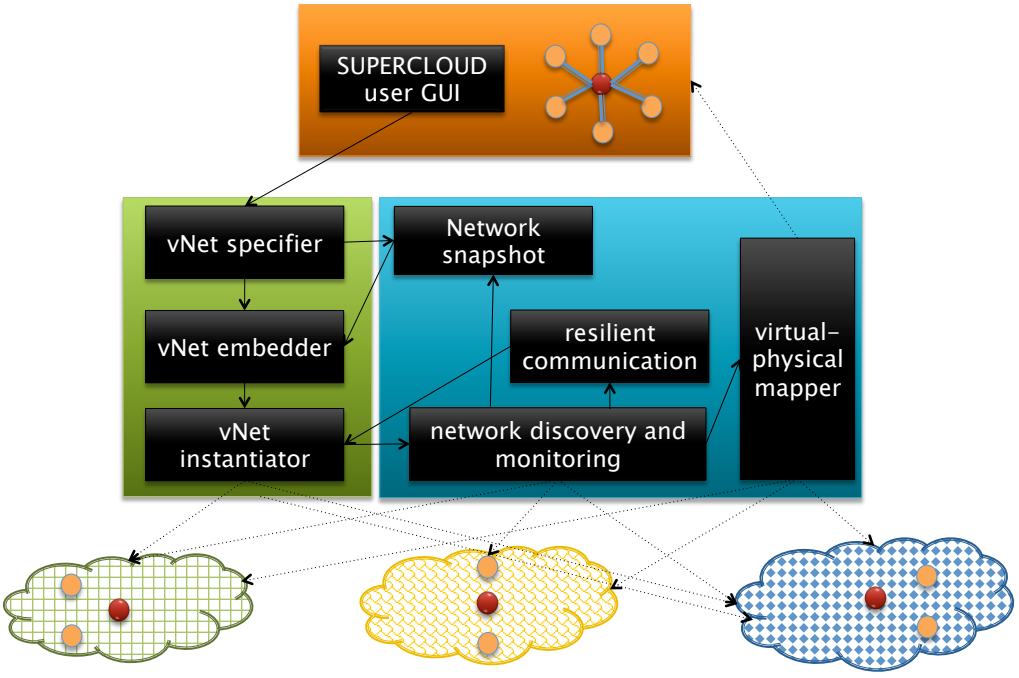


Figure 6.8: Modular architecture of the network hypervisor

The SUPERCLOUD user constructs the required virtual network (vNet) using the hypervisor graphical user interface (a possible sketch of this GUI is shown in Figure 6.9, for a small-scale network). When the user submits its required virtual network a configuration file will be generated with a virtual network request in the vNet specificier.
Then, a Virtual Network Embedding (vNet embedding, or VNE) algorithm performs the mapping of the virtual networks to the specific resources (nodes and links) in the substrate infrastructure. A good amount of literature on VNE algorithms exists that we will leverage on [28]. The typical optimization constraints and objectives of such algorithms are Quality of Service (QoS), economic cost, and survivability. In the context of SUPERCLOUD there is a need to consider an environment with multiple clouds (including inter-cloud communication costs) and security constraints. Examples of the latter include virtual resources that should not be mapped on physical resources that do not comply with its security requirements; physical resources that should not host virtual resources that

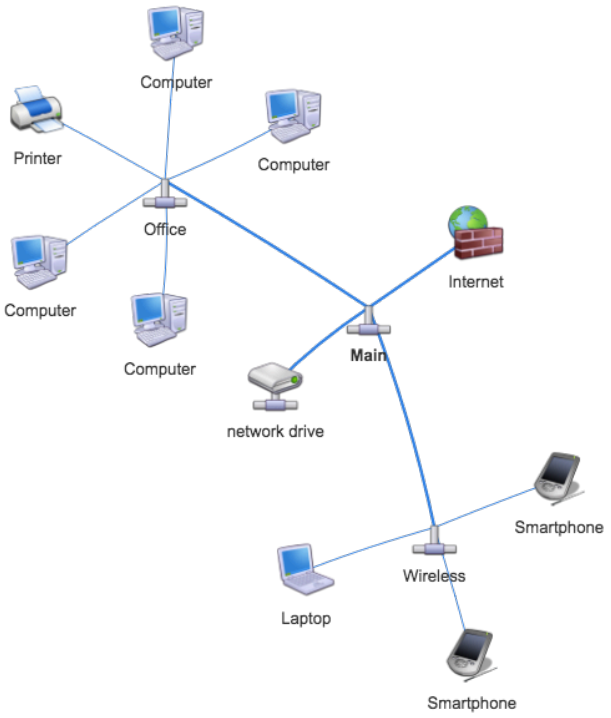


Figure 6.9: A first sketch of the Graphical User Interface

are potentially harmful to its operation; or virtual resources that should not be co-hosted on the same physical resource as another potentially malicious virtual resource. We will explore such constraints in designing our VNE algorithm.
Finally, as the last module of the configuration block, the virtual network is instantiated (Figure 6.10). This includes orchestration tools to set up VMs, using OVSDB to set up tunnels, and using OpenFlow to configure the necessary forwarding tables.

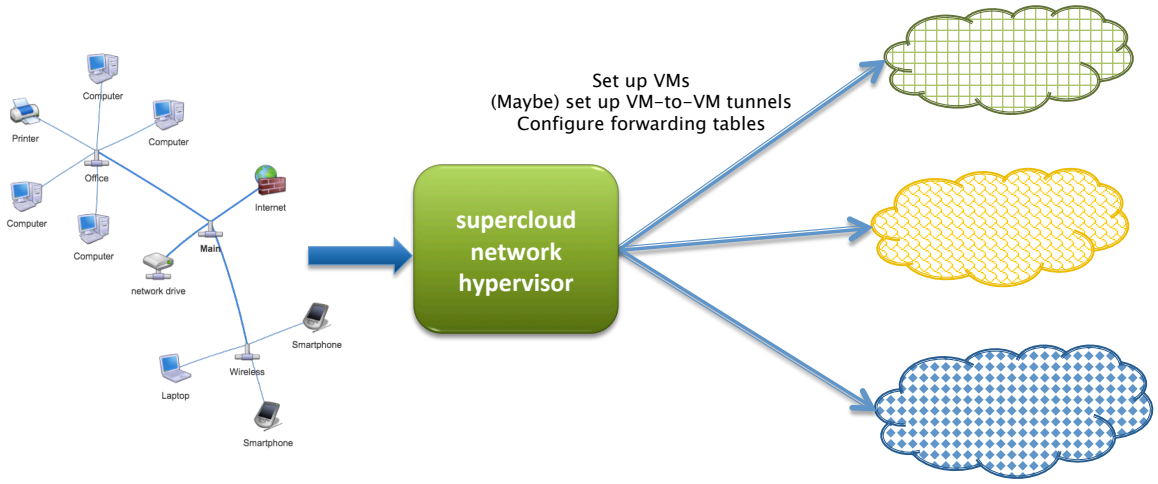


Figure 6.10: Instantiation of the virtual network

The run time block is composed of four modules. The first is the network discovery and monitoring service. This service monitors the topology by capturing all network events (including LLDP messages). As such, it keeps track of links that go up and down, and of switches and hosts that enter and leave the topology. This information is then used by all other modules. The virtual to physical

mapper maps the virtual to physical network events, and vice-versa, in run-time, offering the required isolation and virtualization of topology and addressing. The resilient communication module receives topology change events and reconfigures the topology as needed, to maintain the required level of resilience. Finally, the snapshot module is triggered by a change in the virtual network configuration file (maintained by the vNet specifier module), which in turn triggers a rerun of the VNE algorithm and vNet instantiation.

Chapter 7 Conclusions

In this deliverable we have presented the preliminary architecture of the SUPERCLOUD network virtualization platform:

- We have started by specifying the design requirements of the architecture.
- Then, we presented Software-Defined Networking, the networking paradigm we will use as the basis of our proposal.
- Third, we surveyed recently proposed SDN-based network virtualization platforms. These have offered important insight for the design of our architecture.
- To motivate the need for security management services for the virtualization platform, we then presented a study on the nature and trends of cloud-based security incidents.
- The document culminated in the presentation of the preliminary network virtualization architecture. The focus was on the various techniques we plan to use to fulfill the requirements set forth in our design.

As future work we will implement and evaluate the proposed architecture. As this process progresses we will continuously re-evaluate and improve the platform design. This may include the addition of new modules and the redefinition or refinement of the modules considered in this preliminary version of the architecture.

Bibliography

[1] Best pay-per-install affiliate program reviews. http://pay-per-install.com. Accessed on 31.10.2015.

[2] Financial data stealing malware now on amazon web services cloud. `http://www.securelist.com/en/blog/208188099/Financial_data_stealing_Malware_now_on_Amazon_Web_Services_Cloud`. Accessed on 31.10.2015.

[3] Floodlight. `http://www.projectfloodlight.org/floodlight/`. Accessed on 31.10.2015.

[4] Malware statistics & trends report. http://www.av-test.org/en/statistics/malware/. Accessed on 31.10.2015.

[5] Open Networking Foundation. `https://www.opennetworking.org`. Accessed on 31.10.2015.

[6] VMware NSX. `https://www.vmware.com/products/nsx`. Accessed on 31.10.2015.

[7] A. Abou-El-Kalam, R. Baida, P. Balbiani, S. Benferhat, F. Cuppens, Y. Deswarte, A. Miège, C. Saurel, and G. Trouessin. Organization Based Access Control. In *IEEE 4th International Workshop on Policies for Distributed Systems and Networks*, POLICY '03, 2003.

[8] M. Al-Fares, A. Loukissas, and A. Vahdat. A scalable, commodity data center network architecture. In *Proceedings of the ACM SIGCOMM 2008 Conference on Data Communication*, SIGCOMM '08, 2008.

[9] A. Al-Shabibi, M. De Leenheer, M. Gerola, A. Koshibe, G. Parulkar, E. Salvadori, and B. Snow. OpenVirteX: Make your virtual SDNs programmable. In *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking*, HotSDN '14, 2014.

[10] U. Bayer, P. Milani Comparetti, C. Hlauschek, C. Kruegel, and E. Kirda. Scalable, behavior-based malware clustering. In *NDSS*, volume 9, pages 8–11, 2009.

[11] U. Bayer, C. Kruegel, and E. Kirda. TTAnalyze: A tool for analyzing malware. In *15th European Institute for Computer Antivirus Research Annual Conference*, EICAR '06, 2006.

[12] T. Benson, A. Akella, and D. Maltz. Unraveling the complexity of network management. In *Proceedings of the 6th USENIX Symposium on Networked Systems Design and Implementation*, NSDI '09, 2009.

[13] T. J. Bittman, G. J. Weiss, M. A. Margevicius, and P. Dawson. Magic Quadrant for x86 Server Virtualization Infrastructure. Technical report, Gartner, June 2013.

[14] S. Bleikertz, C. Vogel, and T. Grob. Cloud radar: Near real-time detection of security failures in dynamic virtualized infrastructures. In *Proceedings of ACSAC*, ACSAC '04, 2014.

[15] S. Bleikertz, C. Vogel, T. Grob, and S. Modersheim. Proactive security analysis of changes in virtualized infrastructures. In *Proceedings of ACSAC*, ACSAC '05, 2015.

[16] Z. Bozakov and P. Papadimitriou. AutoSlice: automated and scalable slicing for software-defined networks. In *Proceedings of the 2012 ACM conference on CoNEXT Student Workshop*, CoNEXT Student '12, 2012.

[17] M. Caesar, D. Caldwell, N. Feamster, J. Rexford, A. Shaikh, and J. van der Merwe. Design and implementation of a routing control platform. In *Proceedings of the 2nd USENIX conference on Networked systems design and implementation*, NSDI '05, 2005.

[18] M. Casado. OpenStack and Network Virtualization. `http://blogs.vmware.com/vmware/2013/04/openstack-and-network-virtualization.html`, April 2013. Accessed on 31.10.2015.

[19] M. Casado, M. J. Freedman, J. Pettit, J. Luo, N. Gude, N. McKeown, and S. Shenker. Rethinking enterprise network control. *IEEE/ACM Transactions on Networking*, 17(4):1270–1283, August 2009.

[20] M. Casado, T. Koponen, R. Ramanathan, and S. Shenker. Virtualizing the network forwarding plane. In *Proceedings of the Workshop on Programmable Routers for Extensible Services of Tomorrow*, PRESTO '10, 2010.

[21] D. W. Cearley, D. Scott, J. Skorupa, and T. J. Bittman. Top 10 Technology Trends, 2013: Cloud Computing and Hybrid IT Drive Future IT Models. Technical report, Gartner, 2013.

[22] N. M. M. K. Chowdhury and R. Boutaba. A survey of network virtualization. *Computer Networks*, 54(5):862 – 876, 2010.

[23] B. Davie and J. Gross. A Stateless Transport Tunneling Protocol for Network Virtualization (STT). Internet Draft draft-davie-stt-06 (Informational), April 2014.

[24] J. Dix. Clarifying the role of software-defined networking northbound APIs. `http://www.networkworld.com/news/2013/050213-sherwood-269366.html`, May 2013. Accessed on 31.10.2015.

[25] D. Drutskoy, E. Keller, and J. Rexford. Scalable network virtualization in software-defined networks. *Internet Computing, IEEE*, 17(2):20–27, March 2013.

[26] D. Farinacci, T. Li, S. Hanks, D. Meyer, and P. Traina. Generic Routing Encapsulation (GRE). RFC 2784 (Standards Track), March 2000.

[27] M. C. Ferrer. Zeus in the cloud. `http://community.ca.com/blogs/securityadvisor/archive/2009/12/09/zeus-in-the-cloud.aspx`. Accessed on 31.10.2015.

[28] A. Fischer, J. F. Botero, M. T. Beck, H. de Meer, and X. Hesselbach. Virtual network embedding: A survey. *IEEE Communications Surveys and Tutorials*, 15(4):1888–1906, Fourth 2013.

[29] A. Ford, C. Raiciu, M. Handley, and O. Boneventure. TCP Extensions for Multipath Operation with Multiple Addresses. RFC 6824 (Experimental), January 2013.

[30] N. Foster, A. Guha, M. Reitblatt, A. Story, M.J. Freedman, N.P. Katta, C. Monsanto, J. Reich, J. Rexford, C. Schlesinger, D. Walker, and R. Harrison. Languages for software-defined networks. *Communications Magazine, IEEE*, 51(2):128–134, 2013.

[31] P. Garg and Y. Wang. NVGRE: Network Virtualization Using Generic Routing Encapsulation. RFC 7637 (Informational), September 2015.

[32] K. Greene. MIT Tech Review 10 Breakthrough Technologies: Software-defined Networking. `http://www2.technologyreview.com/article/412194/tr10-software-defined-networking/`, 2009. Accessed on 31.10.2015.

[33] J. Gross, T. Sridhar, P. Garg, C. Wright, I. Ganga, P. Agarwal, K. Duda, D. Dutt, and J. Hudson. Geneve: Generic Network Virtualization Encapsulation. Internet Draft draft-ietf-nvo3-geneve-00 (Informational), May 2015.

[34] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker. NOX: towards an operating system for networks. *SIGCOMM Computer Communication Review*, 38(3):105–110, July 2008.

[35] S. Gutz, A. Story, C. Schlesinger, and N. Foster. Splendid isolation: A slice abstraction for software-defined networks. In *Proceedings of the First Workshop on Hot Topics in Software Defined Networks*, HotSDN '12, 2012.

[36] X. Han, N. Kheir, and D. Balzarotti. The role of cloud services in malicious software: Trends and insights. In *Proceedings of the 12th International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, DIMVA '15, 2015.

[37] B. Heller, S. Seetharaman, P. Mahadevan, Y. Yiakoumis, P. Sharma, S. Banerjee, and N. McKeown. ElasticTree: saving energy in data center networks. In *Proceedings of the 7th USENIX conference on Networked systems design and implementation*, NSDI '10, 2010.

[38] K. J. Higgins. Dropbox, wordpress used as cloud cover in new apt attacks. `http://www.darkreading.com/attacks-breaches/dropbox-wordpress-used-as-cloud-cover-in-new-apt-attacks/d/d-id/1140098?` Accessed on 31.10.2015.

[39] C.-Y. Hong, S. Kandula, R. Mahajan, M. Zhang, V. Gill, M. Nanduri, and R. Wattenhofer. Achieving high utilization with software-driven WAN. In *Proceedings of the ACM SIGCOMM 2013 conference on Data Communication*, SIGCOMM '13, 2013.

[40] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu, J. Zolla, U. Hölzle, S. Stuart, and A. Vahdat. B4: experience with a globally-deployed software defined WAN. In *Proceedings of the ACM SIGCOMM conference*, SIGCOMM '13, 2013.

[41] X. Jin, J. Rexford, and D. Walker. Incremental update for a compositional SDN hypervisor. In *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking*, HotSDN '14, 2014.

[42] R. Ko, P. Jagadpramana, M. Mowbray, S. Pearson, M. Kirchberg, Q. Liang, and B. S. Lee. Trustcloud: A framework for accountability and trust in cloud computing. In *2011 IEEE World Congress on Services*, SERVICES '11, 2011.

[43] R. Koetter and M. Medard. An algebraic approach to network coding. *IEEE/ACM Transactions on Networking*, 11(5):782–795, Oct 2003.

[44] T. Koponen, K. Amidon, P. Balland, M. Casado, A. Chanda, B. Fulton, I. Ganichev, J. Gross, P. Ingram, E. Jackson, A. Lambeth, R. Lenglet, S.-H. Li, A. Padmanabhan, J. Pettit, B. Pfaff, R. Ramanathan, S. Shenker, A. Shieh, J. Stribling, P. Thakkar, D. Wendlandt, A. Yip, and R. Zhang. Network virtualization in multi-tenant datacenters. In *11th USENIX Symposium on Networked Systems Design and Implementation*, NSDI '14, 2014.

[45] T. Koponen, M. Casado, N. Gude, J. Stribling, L. Poutievski, M. Zhu, R. Ramanathan, Y. Iwata, H. Inoue, T. Hama, and S. Shenker. Onix: a distributed control platform for large-scale production networks. In *Proceedings of the 9th USENIX conference on Operating systems design and implementation*, OSDI '10, 2010.

[46] D. Kreutz, F. M. V. Ramos, and P. Verissimo. Towards secure and dependable software-defined networks. In *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*, HotSDN '13, 2013.

[47] D. Kreutz, F. M. V. Ramos, P. E. Verissimo, C.E. Rothenberg, S. Azodolmolky, and S. Uhlig. Software-Defined Networking: a comprehensive survey. *Proceedings of the IEEE*, 103(1):14–76, January 2015.

[48] A. Kumar, S. Jain, U. Naik, A. Raghuraman, N. Kasinadhuni, E. C. Zermeno, C. S. Gunn, J. Ai, B. Carlin, M. Amarandei-Stavila, M. Robin, A. Siganporia, S. Stuart, and A. Vahdat. BwE: Flexible, hierarchical bandwidth allocation for WAN distributed computing. In *Proceedings of the ACM SIGCOMM Conference*, SIGCOMM '15, 2015.

[49] C. Li, B.L. Brech, S. Crowder, D.M. Dias, H. Franke, M. Hogstrom, D. Lindquist, G. Pacifici, S. Pappe, B. Rajaraman, J. Rao, R.P. Ratnaparkhi, R.A. Smith, and M.D. Williams. Software defined environments: An introduction. *IBM Journal of Research and Development*, 58(2):1–11, March 2014.

[50] Zhou Li, Sumayah Alrwais, Yinglian Xie, Fang Yu, and XiaoFeng Wang. Finding the linchpins of the dark web: a study on topologically dedicated hosts on malicious web infrastructures. In *Security and Privacy (SP), 2013 IEEE Symposium on*, pages 112–126. IEEE, 2013.

[51] R. Los, D. Shackleford, and B. Sullivan. The notorious nine cloud computing top threats in 2013. In *Cloud Security Alliance*, 2013.

[52] M. Mahalingam, D. Dutt, K. Duda, P. Agarwal, L. Kreeger, T. Sridhar, M. Bursell, and C. Wright. Virtual eXtensible Local Area Network (VXLAN): A Framework for Overlaying Virtualized Layer 2 Networks over Layer 3 Networks. RFC 7348 (Informational), August 2014.

[53] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. OpenFlow: enabling innovation in campus networks. *SIGCOMM Computer Communication Review*, 38(2):69–74, March 2008.

[54] R. N. Mysore, A. Pamboris, N. Farrington, N. Huang, P. Miri, S. Radhakrishnan, V. Subramanya, and A. Vahdat. Portland: A scalable fault-tolerant layer 2 data center network fabric. In *Proceedings of the ACM SIGCOMM 2009 Conference on Data Communication*, SIGCOMM '09, 2009.

[55] C. Peng, M. Kim, Z. Zhang, and H. Lei. VDN: virtual machine image distribution network for cloud data centers. In *Proceedings of IEEE INFOCOM*, INFOCOM '12, 2012.

[56] R. Perdisci, W. Lee, and N. Feamster. Behavioral clustering of HTTP-based malware and signature generation using malicious network traces. In *USENIX Symposium on Networked Systems Design and Implementation*, NSDI '10, 2010.

[57] R. Perlman. Rbridges: transparent routing. In *Proceedings of IEEE INFOCOM*, INFOCOM '04, 2004.

[58] R. Perlman, D. Eastlake, D. Dutt, S. Gai, and A. Ghanwani. Routing Bridges (RBridges): Base Protocol Specification. RFC 6325 (Standards Track), July 2011.

[59] B. Pfaff and B. Davie. The Open vSwitch Database Management Protocol. RFC 7047 (Informational), December 2013.

[60] B. Pfaff, J. Pettit, T. Koponen, E. Jackson, A. Zhou, J. Rajahalme, J. Gross, A. Wang, J. Stringer, P. Shelar, K. Amidon, and M. Casado. The Design and Implementation of Open vSwitch. In *12th USENIX Symposium on Networked Systems Design and Implementation*, NSDI '15, 2015.

[61] Quantum Community. OpenStack Networking. `http://wiki.openstack.org/Quantum`. Accessed on 31.10.2015.

[62] S. Racherla, D. Cain, S. Irwin, P. Ljungstrom, P. Patil, and A. M. Tarenzio. *Implementing IBM Software Defined Network for Virtual Environments*. IBM RedBooks, May 2014.

[63] F. M. V. Ramos, D. Kreutz, and P. Verissimo. Software-defined networks: On the road to the softwarization of networking. *Cutter IT journal*, May 2015.

[64] J. Reich, C. Monsanto, N. Foster, J. Rexford, and D. Walker. Modular SDN Programming with Pyretic. *USENIX ;login*, 38(5), October 2013.

[65] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman. Role-based access control models. *IEEE Computer*, 29(2):38–47, 1996.

[66] S. Shenker. The Future of Networking, and the Past of Protocols. `http://www.youtube.com/watch?v=YHeyuD89n1Y`, October 2011. Accessed on 31.10.2015.

[67] S. Shenker. Stanford Seminar - Software-Defined Networking at the Crossroads. `http://www.youtube.com/watch?v=WabdXYzCAOU`, June 2013. Accessed on 31.10.2015.

[68] R. Sherwood, G. Gibb, K.-K. Yap, G. Appenzeller, M. Casado, N. McKeown, and G. Parulkar. FlowVisor: A Network Virtualization Layer. Technical report, Deutsche Telekom Inc. R&D Lab, Stanford, Nicira Networks, 2009.

[69] R. Sherwood, G. Gibb, K.-K. Yap, G. Appenzeller, M. Casado, N. McKeown, and G. Parulkar. Can the production network be the testbed? In *Proceedings of the 9th USENIX conference on Operating systems design and implementation*, OSDI '10, 2010.

[70] Symantec. Internet security threat report. Technical Report, 2015.

[71] D. Turull, M. Hidell, and P. Sjödin. Evaluating OpenFlow in libnetvirt. In *The 8th Swedish National Computer Networking Workshop*, NSCNW '12, 2012.

[72] R. Unuchek. GCM in malicious attachments. `http://www.securelist.com/en/blog/8113/GCM_in_malicious_attachments`. Accessed on 31.10.2015.

[73] Y. Wang, H. Wang, A. Mahimkar, R. Alimi, Y. Zhang, L. Qiu, and Y. R. Yang. R3: Resilient routing reconfiguration. In *Proceedings of the ACM SIGCOMM 2010 Conference on Data Communication*, SIGCOMM '10, 2010.

[74] H. Yamanaka, E. Kawai, S. Ishii, and S. Shimojo. AutoVFlow: Autonomous virtualization for wide-area OpenFlow networks. In *Third European Workshop on Software Defined Networks*, EWSDN '14, 2014.

[75] B. Yang, J. Liu, S. Shenker, J. Li, and K. Zheng. Keep forwarding: Towards k-link failure resilient routing. In *Proceedings of IEEE INFOCOM*, INFOCOM '14, 2014.

[76] M. Yu, J. Rexford, J. Sun, Rao S., and N. Feamster. A survey of virtual LAN usage in campus networks. *Communications Magazine, IEEE*, 49(7):98–103, July 2011.

[77] E. Yuan and J. Tong. Attributed Based Access Control (ABAC) for Web Services. In *IEEE International Conference on Web Services*, ICWS '05, 2005.

[78] Z. Zhang, Z. Li, K. Wu, D. Li, H. Li, Y. Peng, and X. Lu. VMThunder: fast provisioning of large-scale virtual machine clusters. *IEEE Transactions on Parallel and Distributed Systems*, 25(12):3328–3338, 2014.