Decentralized SDN Control Plane for a Distributed Cloud-Edge Infrastructure: A Survey

David Espinel Sarmiento, Adrien Lebre, Lucas Nussbaum, Abdelhadi Chari

Decentralized SDN Control Plane for a Distributed Cloud-Edge Infrastructure: A Survey

David Espinel, Adrien Lebre, Lucas Nussbaum, Abdelhadi Chari

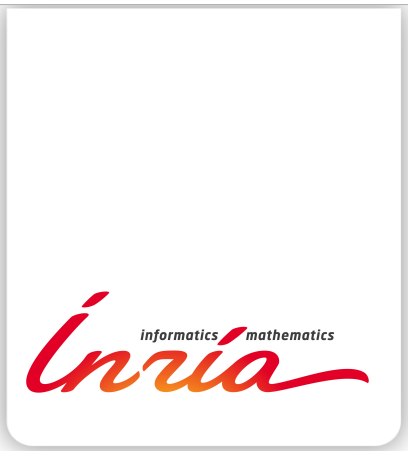
RESEARCH REPORT

N° 9352

June 2020

Project-Teams STACK

ISSN 0249-6399 ISRN INRIA/RR--9352--FR+ENG



Decentralized SDN Control Plane for a Distributed Cloud-Edge Infrastructure: A Survey

David Espinel*†, Adrien Lebre*, Lucas Nussbaum*, Abdelhadi Chari †

Project-Teams STACK

Research Report n° 9352 — June 2020 — 40 pages

Abstract: Today's emerging needs (Internet of Things applications, Network Function Virtualization services, Mobile Edge computing, etc.) are challenging the classic approach of deploying a few large data centers to provide cloud services. A massively distributed Cloud-Edge architecture could better fit the requirements and constraints of these new trends by deploying on-demand infrastructure services in Point-of-Presences within backbone networks. A key feature in this context is the establishment of connectivity among several virtual infrastructure managers in charge of operating, each one, a subset of the infrastructure. After explaining the networking challenges related to distributed Cloud-Edge infrastructures, this article surveys and analyzes the characteristics and limitations of existing technologies in the field of Software Defined Network that could be used to provide the inter-site connectivity feature. This survey is concluded by discussing some research directions.

Key-words: IaaS, SDN, virtualization, distributed networking management, automation

* INRIA
† Orange Labs Lannion

RESEARCH CENTRE
RENNES – BRETAGNE ATLANTIQUE

Campus universitaire de Beaulieu
35042 Rennes Cedex

Plan de contrôle SDN décentralisé pour une intrastructure Cloud-Edge: une étude survey

Résumé : Les besoins émergents d'aujourd'hui (des applications de l'internet des objets, des fonctions réseau virtualisées, le mobile edge computing, etc.) remettent en question l'approche classique consistant à déployer quelques grands centres de données pour fournir des services cloud. Une architecture cloud-edge massivement distribuée pourrait mieux répondre aux exigences et aux contraintes de ces nouvelles tendances en déployant des services d'infrastructure à la demande dans des points de présence au sein de réseaux backbone. Une caractéristique clé dans ce contexte est a mise en place de la connectivité entre plusieurs gestionnaires d'infrastructure virtuelle chargés d'exploiter, chacun, un sous-ensemble de l'infrastructure.

Après avoir expliqué les défis concernant la mise en place des services réseau dans les infrastructures cloud-edge distribuées, cet article examine et analyse les caractéristiques et les limites des technologies existantes dans le domaine du réseau défini par logiciel qui pourraient être utilisées pour fournir la fonctionnalité de connectivité intersites. Ce survey est conclu en discutant de certaines pistes de recherche.

Mots-clés : IaaS, SDN, virtualisation, management distribué du réseau, automatisation

1 Introduction

Internet of Things (IoT) applications, Network Function Virtualization (NFV) services, and Mobile Edge Computing [1] require to deploy IaaS services closer to the end users in order to respect operational requirements. One way to deploy such a distributed cloud infrastructure (DCI) is to extend network points of presence (PoPs) with dedicated servers and to operate them through a cloud-like resource management system [2].

Since building such a DCI resource management system from scratch would be too expensive technically speaking, a few initiatives proposed to build solutions on top of the OpenStack and Kubernetes eco-systems (RedHat DCN [3], StarlingX [4], KubeEdge [5], and kubefed [6] to name a few). These proposals are based either on a centralized approach or a federation of independent Virtual Infrastructure Managers (VIMs).[1] The former lies in operating a DCI as a traditional single data center environment, the key difference being the wide-area networking found between the control and compute nodes. The latter consists in deploying one VIM on each DCI site and federate them through a brokering approach to give the illusion of a single coherent system as promoted by ETSI [7].

Due to frequent isolation risks of one site from the rest of the infrastructure [8], the federated approach presents a significant advantage (each site can continue to operate locally). However, the downside relates to the fact that VIM code does not provide any mechanism to deliver inter-site services. In other words, VIMs have not been designed to peer with other instances to establish inter-site services but rather in a pretty stand-alone way in order to manage a single deployment. Several academic studies investigated how this global vision can be delivered either through a bottom-up or top-down approaches. A bottom-up collaboration aims at revising low-level VIM mechanisms to make them collaborative, using for instance a shared database between all VIM instances [1, 2, 9]. A top-down design implements the collaboration by interacting only with the VIMs'API leveraging for instance a P2P broker [10].

In addition to underlining the inter-site service challenges, these studies enables us to identify key elements a resource management for DCIs should take into account:

- Scalability: A DCI should not be restricted by design to a certain number of VIMs.
- Resiliency: All parts of a DCI should be able to survive to network partitioning issues. In other words, cloud service capabilities should be operational locally when a site is isolated from the rest of the infrastructure.
- Locality awareness: VIMs should have autonomy for local domain management. This implies that locally created data should remain local as much as possible, and only shared with other instances if needed, thus avoiding global knowledge.
- Abstraction and automation: Configuration and instantiation of inter-site services should be kept as simple as possible to allow the deployment and operation of complex scenarios. The management of the involved implementations must be fully automatic and transparent for the users.

In this article, we propose to extend these studies by focusing on the inter-site network service, i.e., the capacity to interconnect virtual networking constructions belonging to several independent VIMs. For instance, in an OpenStack-based DCI (i.e., one OpenStack instance by POP), the networking module, Neutron [11], should be extended to enable the control of both

[1] Unless specified, we used the term VIM in the rest of the article without any distinction between infrastructure management systems such as OpenStack or container ones such as Kubernetes.

intra-PoP and inter-PoP connectivity taking into account the aforementioned key elements. We consider, in particular the following inter-site networking services [12]:

- *Layer 2 network extension*: being able to have a Layer 2 Virtual Network (VN) that spans several VIMs. This is the ability to plug into the same VN, VMs that are deployed in different VIMs.
- *Routing function*: being able to route traffic between a VN A on VIM 1 and a VN B on VIM 2.
- *Traffic filtering, policy and QoS*: being able to enforce traffic policies and QoS rules for traffic between several VIMs.
- *Service Chaining*: Service Function Chaining is the ability to specify a different path for traffic in replacement of the one provided by the shortest path first (SPF) routing decisions. A user needs to be able to deploy a service chaining spanning several VIMs, having the possibility to have parts of the service VMs placed in different VIMs.

Technologies such as Software Defined Networking (SDN), which proposes a decoupling among control and data plane [13], can be leveraged to provide such operations among VIMs [14]. This has been applied for instance to provide centralized control of lower infrastructure layers of WANs (e.g., physical links or fabric routers and switches) in several proposals [15–17], including the well-known Google B4 controller [18], and commonly referred nowadays as Software Defined WAN [19]. Similarly, VIMs specialized in the management of WANs, usually called WAN infrastructure manager (WIM) [20], have been proposed as SDN stacks capable of controlling all the links and connectivity among multiple PoPs [21–23].

The approach in which this survey is interested is called SDN-based cloud computing [24]. Our objective is to study how such services can be delivered in a DCI context leveraging as much as possible existing solutions. To the best of our knowledge, this is the first study that addresses this question. Concretely, our contributions are (i) an analysis of the requirements and challenges raised by the connectivity management in a DCI operated by several VIMs, (ii) a survey of decentralized SDN solutions (analyzing their characteristics and limitations in the context of DCIs), and (iii) a discussion of the research directions in this field.

The rest of this paper is organized as follows. Section 2 defines SDN technologies in general and introduces two SDN-based cloud computing solutions, namely OpenStack and Kubernetes. A review on previous SDN surveys is provided in Section 3. Challenges related to DCI are presented and discussed in Section 4. Definitions of selected criteria used for the survey are explained in Section 5. Section 6 presents characteristics of the studied SDN solutions. The discussion and future work related to the distribution of the SDN control plane for DCIs are presented in Section 7 Finally, Section 8 concludes and discusses future works.

2 SDN and VIMs networking: Background

In this section, we present background elements for readers who are not familiar with the context. First, we remind the main concepts around Software-Defined-Network as they are strong basis for most solutions studied in this survey. Second, we give an overview of how the network is virtualized in OpenStack and Kubernetes, the two de facto open-source solutions to use cloud computing infrastructures.

2.1 *Software-Defined-Network*

The Software-Defined-Network paradigm offers the opportunity to program the control of the network and abstract the underlying infrastructure for applications and network services [13]. It relies on the control and the data plane abstractions. The former corresponds to the programming and managing of the network (i.e., it controls how the routing logic should work) The latter, corresponds to the virtual or physical network infrastructure composed by switches, routers, and other network equipment that are interconnected. These equipment use the rules that have been defined by the control plane to determine how a packet should be processed once it arrives at the device . The idea of the separation of the control plane was introduced by the IETF ForCES Working Group in 2004 [25]. However the SDN paradigm has not been adopted immediately. The work achieved in 2008 around OpenFlow [26] is considered as the first appearance of Software Defined Networks in modern literature. In this initial proposal, the control plane is managed through a centralized software entity called the SDN controller. To communicate with every forwarding device or lower-level components, the controller uses standardized application program interfaces (APIs) called southbound interfaces. In addition to OpenFlow, the most popular southbound APIs are the Cisco's OpFlex ones [27].

Controllers also expose a northbound API, which allows communication among the controller and the higher-level components like management solutions for automation and orchestration. A generic SDN architecture with the aforementioned elements is presented in Figure 1. Overall, an SDN controller abstracts the low-level operations for controlling the hardware, allowing an easy interaction with the control plane, as developers and users can control and monitor the underlying network infrastructure [28].

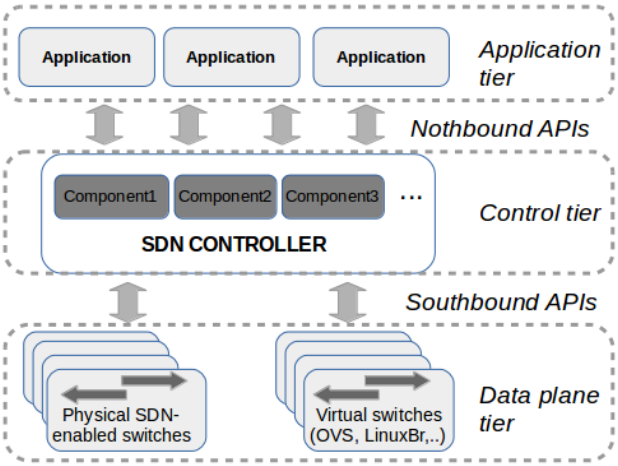


Figure 1: SDN general architecture

One of the possible application domains of the SDN technology lies into the cloud computing paradigm, which exposes to end-users software, applications or virtualized infrastructures in a simple and easy-to-use way [29].

By delivering network programmability, SDN abstractions provide the control and management necessary to cloud computing solutions to expose network resources to end-users. Referred to as SDN-based cloud networking (SDN-clouds or SDN-enabled clouds), this approach enables the configuration and provisioning of virtualized network entities using programmable interfaces

in cloud computing infrastructures. It is used for instance to assure the multi-tenancy needs and the sharing of network resources among end-users [15, 24, 30, 31]. The Neutron OpenStack service described in the next section is a concrete example of such an SDN-based cloud networking service.

2.2 SDN-based Cloud networking services

To illustrate how the network resources are managed in a cloud computing infrastructure using SDN technologies, we discuss in this section the two de facto open-source solutions, namely OpenStack [32] and Kubernetes [33].

2.2.1 *OpenStack Neutron*

Challenge	Key words	Summary
Network information's challenges		
Identifying how information should be shared	**information granularity**	Propose good information sharding strategies
Sharing networking information on-demand and in an efficient manner	**information scope**	Avoid heavy synchronization by contacting only the relevant sites
Facing network disconnections	**information availability**	Continue to operate in cases of network partitioning and be able to recover
Technological challenges		
Standard automatized and distributed interface	**automatized interfaces**	Well-defined and bridged vertical and horizontal interfaces
Support and adaptation of networking technologies	**networking technologies**	Capacity to configure different networking technologies

Table 1: DCI Challenges summary

Neutron is the OpenStack service in charge of delivering "Network connectivity as a Service". It provides on-demand, scalable, and technology-agnostic network services. *Port*, *Network*, and *Subnetwork* are the basic object abstractions offered by Neutron. Those resources, referred to as Core resources are managed by a Core plug-in (the default one being the ML2 Core plug-in). Each abstraction has the same functionality as its physical counterpart: *Network* is an isolated L2 segment, *Subnetwork* is a block of IP version 4 or 6 addresses contained inside a Network, *Port* is a virtual switch connection point used to attach elements like VMs to a virtual network. More network objects can be defined and exposed extending the Neutron API, some examples are *Router*, *floatingIPs*, *L2GWs* [34], *BGP-VPNs* [35] or Virtual Private Network as a Service (VPNaaS) [36] which are known as resource extensions and controller by a Service plug-in.

Conceptually speaking, a single instance of the Neutron module could manage all network resources of a DCI. As long as it is possible to communicate at the IP level between all equipment, Neutron can configure the different network devices and deliver the expected network abstractions. Technically speaking, leveraging a single centralized module cannot satisfy DCI properties (Single Point of Failures, network bottleneck, intermittent connectivity between network equipment, etc.) and the deployment of several collaborative instances (in the worst case,

one per locations) is necessary. Unfortunately, the software architecture of Neutron does not allow collaborations between multiple instances. For instance, the Neutron database belongs to a single Neutron instance, the resources created at one site can only be managed by that Neutron. It is not possible for a distant Neutron to have knowledge nor access to the objects present in a database of another instance. Because information of resources is not shared among Neutrons, the notion of a virtual network spanning different VIMs does not exist today in the Neutron database. Further, operations like identifiers assignation, IP and MAC addresses generation and allocations, DHCP services or security group management are also handled internally at each Neutron instance. Such kind of constraints takes out the possibility to manage Neutron resources in a distributed way.

2.2.2 *Kubernetes networking*

Kubernetes (a.k.a. k8s) is a container orchestration solution. In addition to container deployment mechanisms (i.e., using containerization platforms such as Docker [37] or Linux Containers [38]), it offers scaling in/out, as well as load balancing capabilities. Kubernetes follows a client-server architecture, by default there is a single server that acts as a controlling node and API entry point. This master node manages several worker nodes where the containers are deployed. The main unit of scheduling in Kubernetes is the pod. A pod is a group of containers allocated on the same worker node to share a network stack and other resources.

To provide the networking capabilities for pods, Kubernetes allows end-users to use the default networking configuration or to use a third-party container network interface (CNI) plug-in. While the default k8s networking has a single default network, common to the entire cluster, several CNI plug-in implementations have been proposed to abstract the network within a Kubernetes cluster [39–45]. These plug-ins provide more functionalities than the default one such as firewalls or bandwidth control. Other solutions such as the Kuryr project [46], are wrappers to existing solutions (the Kuryr project exposes Neutron functionalities within Kubernetes). Since every plug-in creates the network in different manners, end-users have the ability to chose the most suitable plug-in depending on their use-cases.

2.2.3 *Summary*

Because VIMs such as OpenStack or Kubernetes have not been designed to peer with other instances, several projects have been investigating how it might be possible to deliver inter-site services. Among existing projects, we can mention Tricircle [47] in the OpenStack ecosystem or Kubernetes Cluster Federation [6]. Both projects aim at exposing multiple Openstack/Kubernetes clusters as a single entity. Unfortunately, these solutions have been designed around a centralized architecture and face important limitations (scalability, network partitions, etc.).

Thus, how to decentralize the management of multiple clusters is an important question that our community must deal with. We propose to initiate the debate by focusing on the OpenStack Neutron service in the following. However, we underline that our study is valuable more generally since it provides an abstract enough analysis of the different challenges related to DCI management.

Before analysing in detail those challenges, we discuss previous works and surveys on SDN technologies. In addition to underlining major studies that might be relevant to better understand SDN concepts, it enables us to position our current work with respect to previous surveys.

3 Reviews on works & surveys on SDN and SDN-based cloud computing

SDN Technology has been a hot topic for the last few years. In that sense several studies have been already published. In this section, we underline the major ones. First we discuss papers that discuss SDN technologies in general. Second, we review SDN-based cloud activities. By introducing these studies, we aim to underline that the use of multi-instance SDN technologies in the context of a DCI has not been analyzed in the literature yet.

In [48] the authors presented a general survey on SDN technologies presenting a taxonomy based on two classifications: physical classification and logical classification. For every one of the classifications, multiple subcategories were presented and explained and the surveyed SDN solutions were placed accordingly to their architectural analysis. The work finished by presenting a list of open questions such as scalability, reliability, consistency, interoperability, and other challenges such as statistics collection and monitoring.

In [49] and [50], the authors focus on the scalability criteria. More precisely, the work done by Karakus et al. [49] provided an analysis of the scalability issues of SDN control plane. the paper surveyed and summarized the characteristics and taxonomy of SDN control plane scalability through two different viewpoints: topology-related and mechanisms-related. The topology-related analysis presents the relation between topology of architectures and some scalability issues related to them. The mechanism-related viewpoint describes the relation between different mechanisms (e.g., parallelization optimization) and scalability issues. The limitation of this work is that the analysis is done by only considering the throughput measured in established flows per second and the flow setup latency.

In [50], Yang et al. provided a scalability comparison among several different types of SDN control plane architectures by doing simulations. To assign a degree of the scalability, the authors proposed to measure the flow setup capabilities and the statistics collection. Although comparisons among controller architectures are made in these two articles, there is no analysis nor mention to the DCI context and related challenges.

Among other available studies in traditional SDN technologies, the work presented in [51] and [52] are probably the most interesting ones with respect to DCI objectives. In their article [51], Blial et al. give an overview on SDN architectures composed by multiple controllers. The study is focused on the distribution methods and the communication systems used by several solutions in order to design and implement SDN solutions able to manage traditional networks. Similarly, the survey [52] discusses some design choices of distributed SDN control planes. it delivers an interesting analysis about the fundamental issues found when trying to decentralize an SDN control plane. These cornerstone problems are scalability, failure, consistency, and privacy. The paper analyses pros and cons of several design choices based on the aforementioned issues. While these two studies provide important information for our analysis, they do not address the cloud computing viewpoint as well as the DCI challenges.

In the field of SDN applied specifically to cloud computing, the works of Azodolmolky et al. [24, 30] provide information about the benefits and potential contributions of SDN technologies applied for the management of cloud computing networking. While these works represent an interesting entry point to analyze the evolution of SDN-based cloud networking, they mostly analyzed the networking protocols and implementations (e.g., VLAN, VXLAN,...) that may be used in order to provide networking federation among a few data centers. More recently, Son et al. [31] presented a taxonomy of SDN-enabled cloud computing works as well as a classification based on their objective (e.g., energy efficiency, performance, virtualization, and security), the method scope (e.g., network-only, and joint network and host), the targeted architecture (e.g., Intra-datacenter network (DCN), and Inter-DCN), the application model (e.g., web application,

map-reduce, and batch processing), the resource configuration (e.g., homogeneous, and heterogeneous), and the evaluation method (e.g., simulation, and empirical). Additional metrics such as data center power optimization, traffic engineering, network virtualization, and security are also used to distinguish the studied solutions. Finally, the paper provides a gap analysis of several aspects of SDN technologies in cloud computing that have not been investigated yet. Among them, one can cite the question related to the extension of cloud computing concepts to the edge of the network (i.e the DCI we envisioned).

To summarize, prior surveys and works neither analyze the challenges of decentralized virtualized networking management in the context of DCIs nor the characteristics (pros/cons) of SDN solutions that could be used to execute this management between multiple instances of the same VIM. The present survey aims to deliver such a contribution.

4 Distributed network control management Challenges

As discussed in Section 2, the control of the network elements of a DCI infrastructure should be performed in a distributed fashion (i.e., with multiple VIM network service that collaborate each other to deliver the same network capabilities across multiple sites). Obviously, decentralizing a controller such as Neutron brings forth new challenges and questions. We choose to divide them into two categories: the ones related to the organization of network information and the ones related to the implementation of the inter-site networking services. These challenges are summarized in Table 1. The key words column is used to introduce the name of the challenges that will be used in the rest of the document. Finally, the term VIM refers to the VIM network service in the following.

4.1 Network information's challenges

Giving the illusion that multiple VIMs behave like a global SDN-based Cloud networking service requires an information exchange. However mitigating as much as possible data communications while being as robust as possible (w.r.t network disconnection or partitioning issues) requires to consider several dimensions as discussed in the following.

4.1.1 Identifying how information should be shared (information granularity)

The first dimension to consider is the organization of the information related to the network elements. As an example, the provisioning of an IP network between two VIMs will require to share information related to the IPs that have been allocated on each VIM. A first approach may consist in sharing the information between the two VIMs each time an IP is allocated to one resource. This way will prevent possible conflict but with an overhead in terms of communications (the global knowledge base is updated each time there is a modification). A second approach would be to split the range of IP addresses with the same Classless Inter Domain Routing (CIDR or network prefix) between the two VIMs at the creation of the network (i.e., each VIM has a subset of the IPs and can allocate them without communicating with other controllers). This way prevents facing IP conflicts even in case of network partitioning without exchanging information each time a new IP is allocated to a particular resource.

Understanding the different structures that are manipulated will enable the definition of different information sharding strategies between multiple VIMs and identify pros and cons for each of them.

Additionally, other elements related to local domain networking management that may be attached to a virtual network as local router gateways, external gateways, DHCP ports, DNS

servers, fixed host routes or floating IPs may not be likely to be shared with remote sites. In consequence, depending on the inter-site service, the granularity of the shared objects information needs to be well specified to avoid conflicts among the networking management entities. If in any case, the joint management of a networking construction is strictly required, the management entities should have the necessary mechanisms to do management coordination in order to provide some kind of data consistency.

4.1.2 Sharing networking information on-demand and in an efficient manner (information scope)

The second dimension to consider is related to the scope of a request. Networking information should stay as local as possible. For instance, network information like MAC/IP addresses of ports and identifiers of a network related to one VIM does not need to be shared with the other VIMs that composed the DCI. Similarly, information related to a Layer 2 network shared between two VIMs as depicted in Figure 2 does not need to be shared with the 3rd VIM. The extension of this Layer 2 network could be done later. That is, only when it will be relevant to extend this network to *VIM 3*.

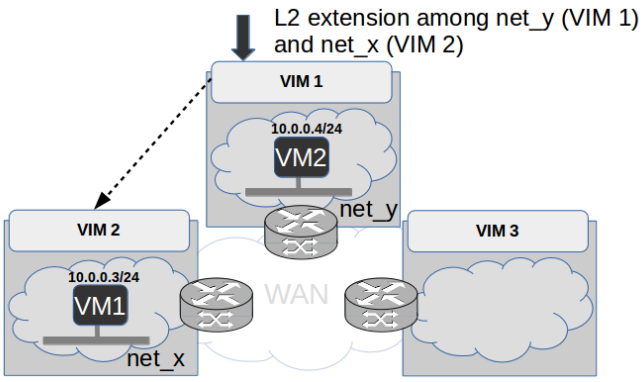


Figure 2: Layer 2 extension Request

Taking into account the scope for each request is critical since sharing information across all VIMs should be avoided due to the heavy synchronization and communication needs. In other words, contacting only the relevant sites for a request will mitigate the network communication overhead and the limitations regarding scalability as well as network disconnections.

Obviously, the information-sharing protocol needs to be fast and reliable to avoid performance penalties that could affect the deployment of the network service.

4.1.3 Facing network disconnections (information availability)

Each VIM should be able to deliver network services even in case of network partitioning issues. Two situations must be considered: (i) the inter-site network resource (for instance a Layer 2 network) has been deployed before the network disconnection and (ii) the provisioning of a new inter-site network resource. In the first case, the isolation of a VIM (for instance *VIM 2* in Figure 3) should not impact the inter-site network elements: *VIM 2* should still be able to assign IPs to VMs using the "local" part of the inter-site Layer 2 network. Meanwhile, *VIM 1* and *VIM 3* should continue to manage inter-site traffic from/to the VMs deployed on this same shared Layer 2 network.

In the second case, because the VIM cannot reach other VIMs due to the network partitioning issue, it is impossible to get information that are mandatory to finalize the provisioning process. The first way to address such an issue is to simply revoke such a request. In this case, the *information availability* challenge is only partially addressed. The second approach is to provide appropriate mechanisms in charge of finalizing the provisioning request only locally (e.g., creating temporary resources). However such an approach implies to integrate mechanisms to recover from a network disconnection.

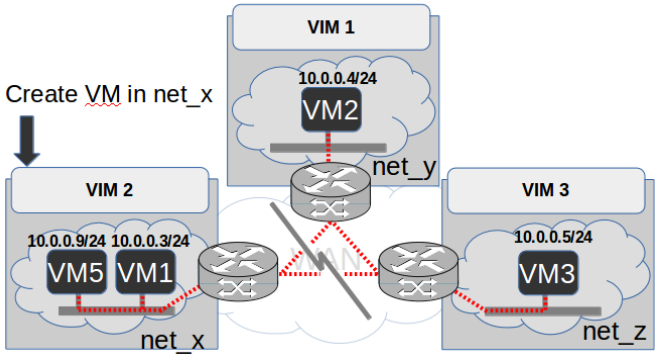


Figure 3: Operate in a local any mode

Depending on the way the resource has been created during the partitioning, the complexity of the re-synchronization procedure may vary.

In the aforementioned scenario, the VIM may do the provisioning of a new VM in *VIM 2* using an IP address already granted to a VM in *VIM 1* or that belongs to another CIDR. Once the network failure is restored, *VIM 1* will face issues to forward traffic to *VIM 2* either because of the overlapping addresses or because there are two different CIDRs.

To satisfy the availability property, the inter-site connectivity management should be able to address such corner cases.

4.2 Technological challenges regarding inter-site networking services

Technological challenges are related to the technical issues that could be presented when trying to implement the DCI networking services.

4.2.1 Standard automatized and distributed interfaces (automatized interfaces)

A first challenge is related to the definition of the vertical and horizontal APIs to allow the provisioning of inter-site services from the end-users viewpoint but also to make the communication/collaboration between the different VIMs possible. This means that the interface which faces the user (user-side or north-side as traffic flows in a vertical way) and the interface which faces other networking services (VIMs-side or east-west-side as traffic flows in a horizontal way) have to be smoothly bridged among them. This integration needs to be done in order to provide the necessary user abstraction and the automation of the VIMs communication process. Consequently, this necessitates the specification and development of well-defined north- and east-west-bound interfaces presenting to the user and to remote instances an abstract enough API with the available networking services and constructions. Thus, how to design efficient APIs

for both north-bound and east-west-bound communication is another problem to address in the case of inter-site connectivity management tools.

Within the framework of OpenStack, Neutron (see Section 2.2) only presents a user oriented interface to provide local services due to its centralized nature.

The Tricircle project [47] partially address this interface automation leveraging a hierarchical architecture where an API gateway node is used as an entry point to a geo-distributed set of OpenStack deployments. Neutron deployments are not aware of the existence of other local Neutrons, but instead always communicate with the Neutron gateway which is also the only interface exposed to the user.

4.2.2 Support and adaptation of networking technologies (networking technologies)

Along with the initial networking information exchanges among VIMs to provide inter-site connectivity (MAC/IP addresses, network identifiers, etc.), the identification of the mechanism to actually do the implementation will be needed. Although there are many existing networking protocols to rely on to do the implementation (VLANs on an interconnection box, BGP-EVPN/IPVPN, VXLAN ids, GRE tunnels, etc.), they will need adaptation in the DCI case. Since the configuration of the networking mechanisms needs to be known by all the participant VIMs in a requested inter-site service, the exchange of additional implementation information will be required among the sites in an automatized way. This automation is required due to the fact that the user should not be aware of how these networking constructions are configured at the low-level implementation. Since a Cloud-Edge infrastructure could scale up to hundreds of sites, manual networking stitch techniques like [34] [35] will be simply not enough.

Depending on the implementation, the solution needs be able to do the reconfiguration of networking services at two different levels:

- At the overlay level which implies the ability to configure virtual forwarding elements like GoBGP instances [53], OpenvSwitch switches [54] or Linux bridges [55]

- At the underlay level which implies the ability to talk or communicate with some physical equipment like the Edge site gateway. As not all physical equipment is OpenFlow-enabled, the possibility to use other protocols may be an advantage when it is necessary to configure heterogeneous components or when internal routes should be exposed to allow traffic forwarding at data plane level.

Additionally, in the context of the challenge described in 4.1.3, the mechanisms used for the implementation need to be capable to reconfigure themselves in order to re-establish the inter-site traffic forwarding.

5 Multi-controller SDN Solutions: Characteristics

In addition to highlight the main challenges that should be addressed by a DCI network service management, our study aims at describing the state of the art of major SDN solutions and the analysis on how each controller may answer the different DCI proposed challenges. In this section, we present an overview of the major differences we identified between the solutions we have studied.

5.1 Architecture

The first point that distinguishes one solution to another is the way controllers are interconnected each other [48–52]. Figure 4 presents the connection topologies we identified during our analysis. We discuss in the following pros and cons of each approach.

Centralized: Architecture presenting a single centralized controller with a global view of the system. It is the simplest and easiest architecture to manage, but at the same time, the less scalable/robust one due to the well-known problem of centralized architectures (SPOF, bottlenecks, network partitioning, etc.).

Hierarchical: Tree-type architecture composed by several layers of controllers. Most solutions present a two-level tree consisting of local controllers and a "root" controller. As the names indicate, local controllers handle local operations such as intra-site routing. At the opposite the "root" controller deals with all inter-site operations. While local controllers only have its own local view and are not aware of the existence of other local controllers, the root controller should maintain a global knowledge of the infrastructure in order to communicate with local controllers each time it is mandatory. While this approach tackles the scalability challenge w.r.t. the centralized approach, it only increases the robustness partially as the root controller is still a centralized point.

Distributed but logically centralized: Architecture where there is one controller per site, managing both intra and inter-site operations. Each time a controller creates or updates a network resource, it broadcasts the modifications to all other controllers. This way enables controllers to maintain an up-to-date copy of the global knowledge, thus acting as a single logical entity. This design stands close to the initial SDN proposition [13] as several controllers share global network information to present themselves as one single controller.

Fully distributed: Architecture similar to the previous one but without communicating all creations/modifications to other controllers. In this approach, locally-created data remains in the instance where it has been created and shared with other instances only when needed. In such a case, explicit communications between controllers are instantiated in order to exchange technical information to establish for instance inter-site services. This way of interconnecting controllers increases the robustness w.r.t network disconnections as a network disconnections or a node failure only impact a subpart of the infrastructure.

Hybrid: Two-layer architecture mixing the distributed and the hierarchical architectures. The control plane consists of several root controllers at the top layer. Each one of the roots manages multiple local controllers who are in charge of their respective site. These root controllers are organized in a distributed fashion, gathering global network state information among them.

5.2 Leader-based operations

When implementing a DCI network service, it is important to consider two kinds of operations: *leaderless vs. leader-based*. Leaderless operations such as creating an only-local network and its sub-networks are "simple" operations that should not lead to network information inconsistencies [52] and thus do not require leadership mechanisms. At the opposite, leader-based operations such as the assignment of an IP in an inter-site network, require a dedicated approach to avoid issues such as IP collisions. For those operations, there should be a leader to take consistent decisions among all controllers. Leaderships can be either given in two ways [56]: in a static

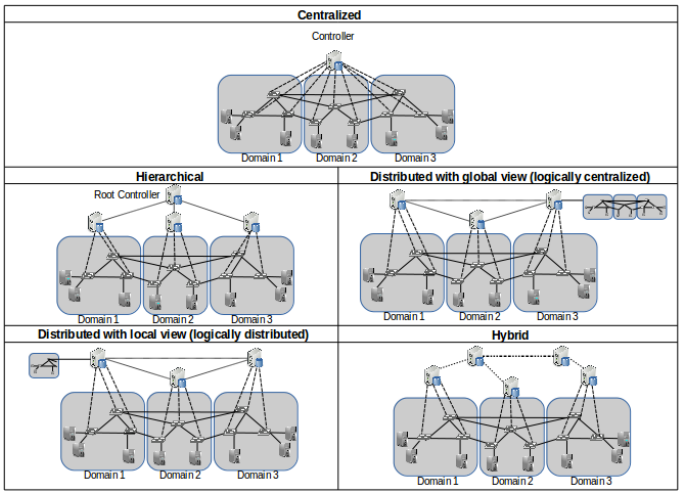


Figure 4: SDN topologies

manner to a controller (i.e., the root node in a hierarchical approach) or by using consensus protocols (i.e., leaderless and leader-based consensus) such as PAXOS [57] or RAFT [58].

Consensus protocols such as the aforementioned ones are used for several tasks such as leader election, group membership, cluster management, service discovery, resource/access management, consistent replication of the master nodes in services, among others [59]. Consensus typically involves multiple instances agreeing on values. Moreover, consensus can be reached when any majority of the instances is available; for instance, a cluster of 5 instances can continue to operate even if two nodes fail. However, applying consensus protocols to a distributed SDN controller may present some problems. In RAFT for instance, network failures can seriously impact the performance of the protocol: in the best case the partitioning may reduce the normal operation time of the protocol; in the worst case, they render RAFT unable to reach consensus by failing to elect a consistent leader [60].

5.3 *Internal communication protocols*

Depending on the selected topology, the communication between controllers occurs either vertically (centralized and hierarchical) or horizontally (distributed). Those communications can be handled through different manners like polling information from others controllers periodically, using a publish/subscribe approach to send notifications in an automatic manner or through explicit communication protocols between controllers.

5.4 Database management system

As largely discussed in Section 4, storing and sharing the state of the DCI network service would be an important challenge. Current solutions rely either on relational or noSQL databases. The former provides important properties to avoid consistency issues but cannot be distributed among geo-distributed resources in an efficient and scalable manner. The latter has been designed to cope with the horizontal scalability challenge but at the cost of losing the strong consistency property.

More generally, the database management system would be a key element of a DCI network

service as it could be used as the means to share information between controllers and thus, eliminates the need for a dedicated communication protocol as discussed in the previous paragraph.

5.5 SDN interoperability and maturity

The studied SDN controllers should be capable of achieving a good performance in heterogeneous and dynamic network environments. For this reason, the capacity to configure different kind of equipment and the maturity of the solution will be explained in this section.

5.5.1 *Network types targeted*

The popularity of virtualization technologies leads to the abstraction of the physical network (a.k.a. the underlay network) into multiple virtual ones (a.k.a. overlay networks).

- *Underlay network*: Is a physical infrastructure that can be deployed in one or several geographical sites. It is composed by a series of active equipment like switches or routers connected among them using Ethernet switching, VLANs, routing functions, among other protocols. Due to the heterogeneity of equipment and protocols, the Underlay network becomes complex and hard to manage, thus affecting the different requirements that must be addressed like scalability, robustness, and high bandwidth.

- *Overlay network*: Virtual network built on top of another network, normally the underlying physical network, and connected by virtual or logical links. Overlay networks help administrators tackle the scalability challenge of the underlay network. For instance, overlay networks leverage the use of encapsulation protocols like VXLAN because of its scalability e.g. VXLAN provides up to 16 million identifiers while VLAN provides 4096 tags.

Because of these two levels of complexity, SDN controllers could be designed to deal with both levels or just one. Obviously, the richer the operations, which are offered by controllers, the more difficult it would be to distribute the DCI network service.

5.5.2 *Supported Southbound protocols*

The reference SDN architecture exposes two kinds of interfaces: Northbound and Southbound. Northbound interfaces reference the protocol communication between the SDN controller and applications or higher layer control programs that may be automation or orchestration tools. Southbound interfaces are used to allow the SDN controller to communicate with the physical/virtual equipment of the network. OpenFlow [61] is an industry standard considered as the de-facto southbound interface protocol. It allows entries to be added and removed to the internal flow-table of switches and potentially routers, so forwarding decisions are based on these flows. In addition to OpenFlow, SDN controllers may use other protocols to configure network components like NETCONF, LISP, XMPP, SNMP, OVSDB, BGP, among others [13]. BGP (Border Gateway Protocol) for example, allows different Autonomous Systems (ASes) to exchange routing and reachability information between edge routers.

More generally, as not all physical equipment is OpenFlow-enabled, the possibility to use others protocols may be an advantage when it is necessary to configure heterogeneous components or when internal routes should be exposed to allow communication at data plane level.

5.5.3 *Readiness Level*

The Technological Readiness Level (TRL) scale is an indicator of the maturity level of particular technology. Due to the complexity of the mechanisms we are dealing with in this survey, it is important to consider the TRL of a technology in order to mitigate as much as possible development efforts. This measurement provides a common understanding of technology status and allows us to establish the status of the surveyed SDN solutions, as not all solutions have the same maturity degree. For this end, the TRL proposed by the European Commission presented in [62] has been used to classify the different solutions we studied.

5.5.4 *Additional considerations*: OpenStack compatibility

As we also propose to take the example of an OpenStack-based system to explain how SDN solutions could be used in a multi-VIM deployment, the capability to integrate with Neutron is introduced as illustrative example. Indeed, some SDN controllers may be able to integrate with Neutron to implement networking services or to add additional functionalities, consuming the Neutron core API or its extensions, and therefore, having a driver to pass network information to the controller.

6 Multi-controller SDN Solutions

Although it would be valuable, delivering a survey of all SDN controller solutions that have been proposed [63, 64] is beyond the scope of this article. We limited our study to the state of the art of major solutions and selected the best candidates that may fit the Distributed Cloud Infrastructure we are investigating. For the sake of clarity, we present the solutions we studied into two categories:

- Network-oriented SDN: solutions that have been designed to provide network programmability to traditional or virtualized network backbones. The controllers gathered in this category have not been designed to provide SDN capabilities for cloud computing networking environments.
- Cloud-oriented SDN: solutions that proposed an SDN way to manage the networking services of cloud computing infrastructures (as explained in Section 2.1). While some of the controllers gathered in this category have been initially designed to manage traditional networks, they propose extensions to provide SDN features within the cloud networking services.

For each selected proposal, we present a qualitative analysis and summarize their characteristics and whether they address the DCI challenges, respectively in Table 2 and Table 3.

6.1 Network-oriented (controllers for SDN domains)

In this first part, we give an overview of the seven SDN solutions we investigated, namely DISCO [65], D-SDN [66], Elasticon [67], FlowBroker [68], HyperFlow [69], Kandoo [70], and Orion [71].

DISCO

DISCO (Distributed SDN Control Plane) relies on the segregation of the infrastructure into distinct groups of elements where each controller is in charge of one group using OpenFlow as

Table 2: SDN solutions characteristics

Proposals	Model					Implementation					Database management system	Interoperability & maturity				Extra consideration
	Centralized (Single) Controller Designs	Distributed Designs				Coordination Strategy		Internal Communication Protocols				Network types targeted		Southbound Protocols	Readiness Level	OpenStack compatibility
		(Flat) Logically centralized	(Flat) Logically distributed	Hierarchical	Hybrid	Leader-based	Leader-less	Among local nodes	With higher layers	Among root nodes		Underlay	Overlay			
Network-oriented solutions																
DISCO			✓				✓	AMQP	-	-	?		✓	OpenFlow & RSVP-like	PoC (TRL 3)	✗
D-SDN				✓		✓		SC-SC Protocol	MC-SC Protocol	-	-		✓	OpenFlow	PoC (TRL 3)	✗
ElastiCon		✓				✓		DB-in/ TCP channel	-	-	NoSQL DB		✓	OpenFlow	PoC (TRL 3)	✗
FlowBroker				✓		✓		-	FlowBroker control channel	?	?		✓	OpenFlow	PoC (TRL 3)	✗
HyperFlow		✓					✓	WheelFS	-	-	WheelFS		✓	OpenFlow	PoC (TRL 3)	✗
Kandoo				✓		✓		-	Simple message channel	-	-		✓	OpenFlow	PoC (TRL 3)	✗
Orion					✓	?	?	Not needed	TCP channel	Pub/Sub	NoSQL DB		✓	OpenFlow	PoC (TRL 3)	✗
Cloud-oriented solutions																
DragonFlow		✓		✓		✓		DB-in	DB-in	DB-in	NoSQL DB/ others		✓	OpenFlow	Demonstrated (TRL 6)	
Onix		✓				✓		NoSQL DB	?	-	SQL DB & NoSQL DB	✓	✓	OpenFlow & BGP	System prototype (TRL 7)	✗
ONOS		✓				✓		DB-in	-	-	Atomix (NoSQL framework)	✓	✓	OpenFlow, NetConf& others	Proven system (TRL 9)	
ODL (Fed)	✓		✓				✓	AMQP	-	-	In-memory	✓	✓	OpenFlow, BGP & others	Proven system (TRL 9)	
Tungsten	✓			✓		✓		BGP	IFMAP	DB-in	NoSQL DB	✓	✓	XMPP, BGP & others	Proven system (TRL 9)	

Proposals	Organization of network information			Inter-site networking services implementation	
	Information granularity	Information scope	Information availability	Automatized interfaces	Networking technologies
Network-oriented solutions					
DISCO	✓	✓	✓	✓	✗
D-SDN	✓	~	?	✗	✗
ElastiCon	~	?	?	~	✗
FlowBroker	✗	✗	✓	✗	✗
HyperFlow	✓	~	~	~	✗
Kandoo	✗	✗	✓	✗	✗
Orion	✗	✗	?	✗	✗
Cloud-oriented solutions					
DragonFlow	~	?	?	~	~
Onix	~	?	?	~	~
ONOS	~	?	?	~	✓
ODL (Fed)	✓	✓	~	✓	~
Tungsten	✗	✗	?	✗	✓

[1] ✓Challenge completely addressed.
[2] ~ Challenge partially addressed.
[3] ✗Challenge not addressed.
[4] ? Undefined.

Table 3: SDN solutions challenges addressing

Control plane protocol. Each controller has an *intra-domain* (or intra-group) part that provides local operations like managing virtual switches, and an *inter-domain* (or inter-group) part that manages communication with other DISCO controllers to make reservations, topology state modifications or monitoring tasks. For the communication between controllers, DISCO relies on an Advanced Message Queuing Protocol (AMQP) message-oriented communication bus where every controller has at the same time an AMQP server and a client. The central component of every controller is the database where all intra- and inter-domain information is stored. We underline that there is not specific information on how the database actually works in the article that presents the DISCO solution.

DISCO can be considered to have a fully distributed design because every local controller stores information of its own SDN domain only, and, establish inter-domain communication with other controllers to provide end-to-end services only if needed. DISCO controllers do not act as a centralized entity and instead work as independent entities peering among them. It has a leader-less coordination because of its logically distributed condition (each controller is in charge of a subgroup so there is no possible conflict). DISCO's evaluations have been performed on a proof of concept. For this reason, we assigned a TRL of 3 to DISCO.

Addressing the challenges

- Information granularity: Addressed - due to the segregation of the infrastructure into distinct groups.
- Information scope: Addressed - thanks to its per-group segregation. When an inter-domain forwarding path is requested, DISCO controllers use the communication channel to only contact the relevant sites for the request. Thus, avoiding global information sharing.
- Information availability: Addressed - in case of network disconnections, each controller would be able to provide intra-domain forwarding. Besides, controllers that can contact each other could continue to deliver inter-site forwarding. Finally, a recovery mode is partially provided given that disconnected sites only need to contact the remote communication channels to retake the inter-domain forwarding service when the connectivity is reestablished. As aforementioned, we underline that due to its implementation and the information that is manipulated, DISCO is conflict-less. This make the recovery process rather simple.

- Automatized interfaces: Addressed - thanks to the bridge presented among the northbound and east-west interfaces to do inter-controller communication.
- Networking technologies: Not addressed since it does not integrate other networking technologies aside of OpenFlow.

D-SDN

D-SDN (Decentralized-SDN) distributes the SDN control into a hierarchy of controllers; i.e., Main Controllers (MCs) and Secondary Controllers (SCs), using OpenFlow as control plane protocol. Similarly to DISCO, SDN devices are organized by groups and assigned to one MC. One group is then divided into subgroups managed by one SC (each SC requests one MC to take the control over a subgroup of SDN devices). We underline that the current article does not give sufficient details regarding how states are stored within Main and Secondary Controllers. The paper mainly discusses two protocols. The first one is related to communications between SCs and MCs using *D-SDN's MC-SC* protocol for control delegation. The second one, entitled *D-SDN's SC-SC* has been developed to deal with fail-over scenarios. The main idea is to have replicas of SCs in order to cope with network or node failures.

As stated in the proposition, D-SDN has a hierarchical design: the MC could be seen as a root controller and SCs as local controllers. It has a leader-based coordination with MC being the natural leader in the hierarchy. As D-SDN is presented as a proof of concept, we defined a TRL of 3.

- Information granularity: Addressed - due to the segregation of the infrastructure elements into distinct groups.
- Information scope: Not addressed - the MC gathers global knowledge, and the communication between SC appears just for fault tolerance aspects.
- Information availability: Undefined - in case of disconnection, SCs controllers can continue to provide forwarding within its local domain at the first sight. However, the article does not specify how the MC deals with such a disconnection. Besides, the controller does not provide any type of recovery method as D-SDN does not take into account network partitioning issues.
- Automatized interfaces: Not addressed - D-SDN proposes an interface for SC-SC communication only for fault tolerance issues. Moreover, there is not information regarding MC-MC communication patterns.
- Networking technologies: Not addressed - since it does not integrate any other networking technologies nor the capacity to provide inter-group service deployment.

ElastiCon

Elasticon (i.e., elastic controller) is an SDN controller composed by a pool of controllers. The pool can be expanded or shrunk according to the size of the infrastructure to operate. Each controller within the pool is in charge of a sub-set of the SDN domain using OpenFlow as control plane protocol. The elasticity of the pool varies according to a load window that evolves over time. Based on the computed value, a centralized module triggers reconfigurations of the pool like migrating switches among controllers or adding/removing a controller. While decisions are made centrally, it is noteworthy to mention that actions are performed by the controllers. To do so,

each controller maintains a TCP channel with every other one creating a full mesh. This protocol enables controllers to coordinate themselves if need be. The states of Elasticon are shared through the Hazelcast distributed data store [72], which can be accessed by all controllers. The use of a shared back-end by the pool gives as result a physically distributed but logically centralized design. As stated in Elasticon's work, the solution has been implemented as a prototype, and thus a TRL of 3 has been assigned to it.

- Information granularity: Partially addressed - Elasticon has been designed to distribute the control of infrastructure over several controllers. If the Hazelcast data store can be deployed across several sites, it is possible to envision to distribute also the pool of controllers between the different sites. By accessing the same database, controllers will be able to add information to the database and fetch the others' information to establish inter-site services. However, the consistency of the data store might be another issue to deal with.
- Information scope: Undefined - it is linked to the database capabilities (in this case, to the way the Hazelcast data store shards the information across the different sites of the infrastructure). However, it is noteworthy to mention that most advanced database systems such as CockroachDB only favor data-locality across several geo-distributed sites partially.
- Information availability: Undefined - similarly to the previous challenge, it is linked to way the database services deals with network partitioning issues. In other words, intra/inter domain forwarding paths that have been previously established should go on theoretically (network equipment have been already configured). Only the recovery mechanism to the DB is unclear.
- Automatized interfaces: Partially addressed - because each controller already has a TCP channel to communicate with the other controllers. However, this communication channel is only used for coordination purposes.
- Networking technologies: Not addressed - since it only operates in OpenFlow-based scenarios.

FlowBroker

FlowBroker is a two layers architecture using OpenFlow as control plane protocol. It is composed by a series of broker agents and semi-autonomous controllers. The broker agents are located at the higher layer. They are in charge of maintaining a global view of the network by collecting SDN domain-specific network state information from the semi-autonomous controllers, which are deployed at the bottom layer. Semi-autonomous controllers do not communicate among them, so they are not aware of the existence of other controllers in the network. These controllers are only aware of interfaces in the controlled switches, thus, providing local-domain forwarding. By maintaining a global view, the broker agents can define how semi-autonomous controllers should establish flows to enable inter-domain path forwarding.

FlowBroker presents a hierarchical design clearly with broker agents acting as root controllers and semi-autonomous domain controllers as local controllers. Although semi-autonomous controllers can establish forwarding paths inside their own domain, the communication with the broker agents is mandatory for inter-domain forwarding. Because of this reason, FlowBroker presents a leader-based coordination, where brokers act as leaders. We underline, however, that there is not any information describing how the information is shared between the different brokers.

Regarding the maturity, we assigned a TRL of 3 to FlowBroker because only a proof-of-concept has been implemented.

Addressing the challenges

- Information granularity: Not addressed - the segregation into semi-autonomous controllers enables the efficient sharding of the information per site. However, the global view of the information that is maintained by the brokers does not enable the validation of this property.
- Information scope: Not addressed - although the global view maintained by each broker allows them to contact only the semi-autonomous controllers that are involved into the inter-service creation, the result of each operation is forwarded to each broker in order to maintain the global view up-to-date.
- Information availability: Addressed - as aforementioned, semi-autonomous controllers can continue to provide local-domain forwarding without the need of the brokers. In the hypothetical case of a network disconnection and the subsequent reconnection, interconnected controllers can still forward the traffic among them. Actually, they only need to contact brokers in order to request inter-site forwarding configuration. Once the configuration of network equipment has been done, controllers do not need to communicate with brokers. Regarding the loss of connectivity with brokers, the recovery process is quite simple because the information shared between all brokers and semi-autonomous controllers is conflict-less.
- Automatized interfaces: Not addressed - because semi-autonomous controllers do not have an east-west interface to communicate among them, but only communicate with brokers. Moreover, the way brokers exchange network knowledge to gather global network view is not discussed.
- Networking technologies: Not addressed - since its use is only intended with OpenFlow protocol.

HyperFlow

Hyperflow is an SDN NOX-based [73] multi-controllers using OpenFlow as control plane protocol. The publish/subscribe message paradigm is used to allow controllers to share global network state information, and is implemented using WheelFS [74]. Each controller subscribes to three channels: data channel, control channel, and its own channel. Events of local network domains that may be of general interest are published in the data channel. In this way, information propagates to all controllers allowing them to build the global view. Controller to controller communications are possible by publishing into the target's channel. A heartbeat is published by every controller in the control channel to notify about its presence on the network.

As the global networking information is shared by all participant controllers, the controller topology presents a physically distributed but logically centralized design. Every controller manages its own domain. In case of network partitions, traffic forwarding can continue inside each controller domain and between the controllers that can contact each other. However, the dependency with respect to WheelFS is not discussed. In other words, the behavior of a controller that cannot contact WheelFS is undefined. More generally, the publish/subscribe paradigm enables Hyperflow to be leader-less. As this proposition has been implemented as a proof-of-concept, a TRL of 3 has been assigned to HyperFlow.

Addressing the challenges

- Information granularity: Addressed - thanks to WheelFS, it is possible to deploy one controller per site. Each one uses WheelsFS to share networking information in order to create

inter-domain forwarding paths.

- Information scope: Partially addressed - HyperFlow presents both a general information data channel and the possibility to communicate directly to specific controllers using their respective channel. Unfortunately, the paper does not clarify whether the establishment of inter-site forwarding is done by contacting the relevant controllers or if instead the general channel is used. In the former case, the exchange is efficient, in the latter, the information will be shared through all controllers.

- Information availability: Partially addressed - in case of network partitioning every controller can continue to serve their local forwarding requests. Regarding inter-forwarding, the dependency w.r.t. to WheelFS is unclear. Theoretically speaking, inter-forwarding channels should survive to disconnections (at least among the controllers that can interact). Moreover, WheelFS provides a recovery method to deal with network partitioning issues. Such a feature should enable controllers to requests new inter-forwarding paths after disconnections without requiring to implement specific recovery mechanisms. Similarly to previous solutions, this is possible because the information shared through WheelFS is conflict-less.

- Automatized interfaces: Partially addressed - since WheelFS is used as both communication and storage utility among controllers. Thus, it is used as the east-west interface. However, HyperFlow's authors underlined that the main disadvantage of the solution is the use of WheelsFS: WheelsFS can only deal with a small number of events, leading to performance penalties in cases where it is used as a general communication publish/subscribe tool among controllers.

- Networking technologies: Not addressed - since it does not integrate other networking technologies besides OpenFlow.

Kandoo

Kandoo is a muti-controller SDN solution built around a hierarchy of controllers and using OpenFlow as control plane protocol. At the low level, local-domain controllers are in charge of managing a set of SDN switches and processing local traffic demands. At the high-level, the single root controller gathers network state information to deal with inter-domain traffic among the local domains. The authors of the Kandoo proposal claim that there are only a few inter-domain forwarding requests and that a single root controller is large enough to deal with. Regarding the local controllers, they do not know about the existence of the others, thus only communicate with the root controller using a simple message channel to request the establishment of inter-domain flows. Unfortunately, the authors did not give sufficient information to understand how this channel works and how the inter-domain flows are set up.

By its two level hierarchical design, Kandoo presents a leader-based coordination (the root controller being the natural leader of the architecture). As the solution had been implemented as a proof-of-concept, a TRL of 3 has been assigned to Kandoo.

Addressing the challenges

- Information granularity: Not addressed - the root controller is used to get information to do inter-domain traffic forwarding, thus gathering the global network view.

- Information scope: Not addressed - similarly to the previous challenge, there is no direct communication between controllers: the single root controller is aware about all inter-domain requests.
- Information availability: Addressed - similarly to FlowBroker solution, the root controller is only required to configure the inter-domain traffic. Once network equipment have been set up, there is no need to communicate with the root controller. The recovery process between local controllers and the root is simple: it consists in just recontacting the root once the network connectivity re-appears (similarly to FlowBroker is conflict-less).
- Automatized interfaces: Not addressed - there is not an east-west interface to communicate among local controllers.
- Networking technologies: Not addressed - since Kandoo does not implement other protocols besides OpenFlow.

Orion

Orion is presented as a hybrid SDN proposition using OpenFlow as control plane protocol. The infrastructure is divided into domains that are then divided into areas. Orion leverages area controllers and domain controllers. Area controllers are in charge of managing a sub-set of SDN switches and establish intra-area routing. Domain controllers, at the top layer, Tare in charge of synchronizing global abstracted network information among all domain controllers, and to establish inter-area routing paths for their managed area controllers. Synchronization of network states between domain controllers is done using a scalable NoSQL database. Moreover, a publish/subscribe mechanism is used to allow domain controllers to demand the establishment of inter-area flows among them. Finally, it is noteworthy to mention that area controllers are not aware about the existence of other area controllers and only communicate with its respective domain controller. This communication is done via a simple TCP channel.

Orion is the only solution that presents a hybrid design: each domain follows a two-level hierarchy and all domain controllers are arranged in a P2P way, using a NoSQL database to share information between each other. Unfortunately the paper does not give details regarding the NoSQL database nor the coordination protocol among the domain controllers. Hence, it is not clear whether Orion uses a leader-based coordination in its P2P model. As the solution had been implemented as a proof-of-concept, a TRL of 3 has been assigned to Orion.

Addressing the challenges

- Information granularity: Not addressed - although the infrastructure is divided into domains (each domain controller maintains its own view of the information), each area controller should notify its domain controller about all changes that occur at the low level.
- Information scope: Not addressed - first, area controllers cannot contact directly other controllers to set up inter-site forwarding services and second, we do not know how information is shared between domain controllers (i.e., it is related to the database system, see Elasticon for instance).
- Information availability: Undefined - in case of network disconnections area controllers can continue to forward intra-domain traffic and inter-domain traffic on paths that have been previously established. In other words, domain controllers are used only for inter-domain path forwarding establishments. In case of a network disconnection, the area controller

only needs to reconnect to its domain controller when needed and when the connection reappears. There is no need for a specific recovery protocol because the information shared between area controllers and their respective domain controller is conflict-less. Only the recovery mechanism related to the DB that is used to share information among domain controllers is unclear.

- Automatized interfaces: Not addressed - due to the fact that local controllers do not present an east-west interfaces to communicate among them.
- Networking technologies: Not addressed - since it does not integrate other networking technologies aside of OpenFlow.

6.2 Cloud-Oriented

In this second part, we present the five solutions we selected from the cloud computing area, namely DragonFlow [75], Onix [76], ONOS [77], ODL [78], and Tungsten [79].

DragonFlow

DragonFlow is an SDN controller for the OpenStack ecosystem, i.e., it implements the Neutron API and thus can replace the default Neutron implementation (see Section 2.2). From the software architecture, DragonFlow relies on a centralized server (i.e., the Neutron server) and local controllers deployed on each compute node of the infrastructure. Each local controller manages a virtual switch providing switching, routing and DHCP capabilities using entirely OpenFlow. A DragonFlow ML2 mechanism driver and a DragonFlow service plug-in are activated in Neutron Server in order to provide system network information to all local controllers. Communication between the plug-ins at the Neutron server side and local controllers is done via a pluggable distributed database (currently supporting OVSDB [80], RAMCloud [81], Cassandra [82], and using ETCD [83] as default back-end).

Local controllers periodically fetch all information of network state through this database and update virtual switches, routes, etc. accordingly.

By maintaining a global knowledge of the network elements through its distributed database, DragonFlow can be considered as a distributed but logically centralized controller (see Section 5.1) at the first sight. However, the fact that there is a root controller (i.e., the Neutron server side) in charge of the management layer (i.e., updating configuration states in the distributed database) and local controllers that implement the control plane makes DragonFlow more a hierarchical solution than a distributed one. In other words, for all leader-based operations, the Neutron plug-in deployed at the server side acts as the leader. In conclusion, although DragonFlow is presented as a distributed SDN controller, its design does not allow the management of a geo-distributed infrastructure (i.e. composed of multiple SDN controllers).

From the maturity view point and according to its activity, we believe DragonFlow has reached a TLR 6. Initially supported by Huawei, the project is rather inactive right now.

Addressing the challenges

- Information granularity: Partially addressed - similarly to Elasticon, if the distributed database service can be deployed across several sites, we can envision an infrastructure composed of several DragonFlow Neutron plug-in. Each one will add information to the database and all local controllers will be capable to fetch the necessary information to provide end-to-end services.

- Information scope: Undefined - it is linked to the way the distributed database system shards the information across the different sites of the infrastructure.

- Information availability: Undefined - similarly to the previous challenge and to the Elasticon solution, it is linked to way the distributed database services deals with network partitioning issues.

- Automatized interfaces: Partially addressed - DragonFlow controllers do not present an east-west interface to communicate with remote sites. Instead, the distributed database is used as communication tool.

- Networking technologies: Partially addressed - the controller incorporates the adaptation and reconfiguration of networking services but it lacks the heterogeneity of networking protocols. Currently DragonFlow does not support BGP dynamic routing [84].

Onix

Onix is a multi-controller SDN platform. In other words, Onix presents several building blocks to develop network services in charge of operating either overlay (using OpenFlow by default) or underlay (using BGP if needed) networks.

Onix's architecture consists of multiple controller instances that share information through a data store called Network Information Base (NIB). The infrastructure is divided into domains, each domain being managed by one instance. Depending on durability and consistency, a network service may use a specific database to implement the NIB module. If durability and strong consistency are required, a replicated transactional SQL database should be used among the instances. Otherwise, it is possible to use any kind of NoSQL system.

Regarding coordination aspects, the system leverages ZooKeeper [85] to deal with instance failures (using the Zookeeper Atomic Broadcast protocol for leader election). The responsibility of the SDN equipment is then determined among the controllers.

By using multiple controllers, and a global network database, the Onix architecture corresponds to a physically distributed but logically centralized one.

As Onix was built as a basis for Nicira's SDN products but was not really a commercial product, a TRL of 7 has been assigned to it.

Finally, the Onix platform integrates some applications including the management of multi-tenant virtualized DCs. This service allows the creation of tenant-specific Layer 2 networks establishing tunnels among the hypervisors hosting VMs in one single deployment. However, this module work in a stand-alone mode and does not interact with the OpenStack Neutron service.

Addressing the challenges

- Information granularity: Partially addressed - similarly to solutions such as Elasticon or DragonFlow, it is related to the database used to share the information between the instances.

- Information scope: Undefined - similarly to the previous challenge, it is related to the database. In case of strong consistency, the information should be synchronized across all instances. in case of a NoSQL system, it depends on how the DB shards the information across the different instances.

- Information availability: Undefined - established services can go on and disconnected sites can continue to operate in isolated mode. The main issue is related to the NIB that should provide the necessary consistency algorithms to allow recovery in case of network disconnection.
- Automatized interfaces: Partially addressed - similarly to DragonFlow, the use of distributed DB to share information among instances can be seen as an east-west interface allowing communication among controllers.
- Networking technologies: Partially addressed - the solution has been designed to use several networking technologies and protocols. Although the initial Onix proposition only supported OpenFlow, Onix design does not impose a particular southbound protocol but rather the use of the NIB as an abstraction entity for network elements.

ONOS

ONOS (Open Network Operating System) is a modular and distributed SDN framework consisting of several network applications build on top of Apache Karaf OSGi container [86]. It supports the use of multiple control plane protocols like OpenFlow, NetConf, and others. ONOS has been created for overlay and underlay networks of service providers. Network states' information is stored using the Atomix database [87], a NoSQL framework developed for ONOS, which is also used for coordination tasks among controllers.

Similarly to other proposals, the infrastructure is divided into domains with one controller per domain. Considering the shared back end and the multiple controller instances, ONOS presents a physically distributed but logically centralized design. As aforementioned, ONOS has a leader-based coordination approach, leveraging the Atomix DB (more precisely, it uses the RAFT algorithm). Considering that ONOS is one of the most popular SDN open-source controllers and is used by several key actors in telecommunications [88], a TRL of 9 has been assigned to ONOS.

Finally, the modular design of ONOS allows the implementation of the Neutron API. Concretely, there are three applications, which consume Neutron API and provide ML2 drivers and Services plug-ins: SONA (Simplified Overlay Network Architecture), VTN (Virtual Tenant Network), and CORD (Central Office Re-architected as a Datacenter) VTN. Each application has been designed with different targets [89,90]. SONA provides an ML2 driver and a L3 service plug-in implementation. VTN provides service function chaining capabilities. CORD VTN extends VTN with its own interfaces for switching and routing configuration [91].

Addressing the challenges

- Information granularity: Partially addressed - similarly to previous solutions that are composed around several instances and a global share database.
- Information scope: Undefined - it is linked to the way the Atomix database system shards the information across the different instances.
- Information availability: Undefined - similarly to the previous challenge it is linked to the Atomix system.
- Automatized interfaces: Partially addressed - ONOS controllers use the Atomix framework for coordination tasks among controllers and to communicate among them.
- Networking technologies: Addressed - ONOS includes several networking technologies.

OpenDayLight

OpenDayLight (ODL) is a modular SDN platform supporting a wide range of protocols such as OpenFlow, OVSDB, NETCONF, BGP, among others. Originally, OpenDayLight was developed as a centralized controller to merge legacy networks with SDN in Data centers, but its modularity allows users to build their own SDN controller to fit specific needs [92]. The controller internal architecture is composed by three layers: The southbound interface, which enables the communication with network devices. The Service Adaptation Layer, which adapts the southbound plug-ins functions to higher-level application/service functions, and finally, the northbound interface, which provides the controller's API to applications or orchestration tools. Network states are stored through a tree structure using a dedicated in-memory data store (i.e., developed for ODL). While the default implementation of ODL can be used in cluster mode for redundancy and high availability, its modularity allows the introduction of features aiming to allow different instances of the controller to peer among them like the SDNi [93] or the more recent Federation [94] projects. ODL Federation service facilitates the exchange of state information between multiple OpenDayLight instances, it relies on AMQP to send and receive messages to/from other instances. A controller could be at the same time producer and consumer.

The Federation project of ODL corresponds to a physical and logical distributed design (each instance maintains its own view of the system). Moreover it is a leader-less coordination approach because there is a flat on-demand communication between controllers and no leader is needed for these exchanges.

The modularity of the controller allows multiple projects to implement the Neutron API. For instance, ODL comes with the OpenStack Neutron API application. This application provides the abstractions that are mandatory for the implementation of the Neutron API inside the controller. Among those implementations we found: Virtual Tenant Network Manager (VTN), Group Based Policy (GBP), and OVSDB-based Network Virtualization Services (NetVirt) [95].

By leveraging the Federation and NetVirt projects, it is possible to create virtual network resources spreading across several OpenStack instances. When the Federation manager receives a request to create an inter-site service between two OpenStack instances, it realizes the interconnection at the ODL level (i.e., creating shadow elements, etc.) and performs the matching with the OpenStack Neutron resources on the different sites. Although this enables to interconnect multiple instances of OpenStack, it is noteworthy to mention that Neutron instances remain unconscious of the information shared at the ODL level. In other words, there is no coordination mechanism that will prevent overlapping information at the Neutron level. This is rather critical as it may lead to consistency issues where an IP for instance can be allocated on each site without triggering any notification.

Since ODL is a community leader and industry supported framework presented in several industrial deployment and continuous development, a TRL of 9 has been assigned to ODL [96].

Addressing the challenges

- Information granularity: Addressed - through the Federation project, it is possible to leverage several controllers to operate an infrastructure (each controller maintains its own view).
- Information scope: Addressed - each controller can interact with another one by using AMQP. In other words, there is not any information that is shared between controllers unless needed.
- Information availability: Partially addressed - in case of network disconnection, ODL instances can satisfy local networking services (including the VIM ones). At the same time, the non-disconnected controllers can continue to provide the inter-site services. Since the

inter-site services are proposed outside the knowledge of the VIM networking module, ODL assumes that there are no conflicts between networking objects when establishing the service. Actually, ODL can not provide a recovery method in case of incoherence since it is not the entity in charge of the networking information management. This is an important flaw for the controller when it needs to recovery from networking disconnections.

- Automatized interfaces: Addressed - thanks to the use of AMQP as east-west interface among the controllers.
- Networking technologies: Addressed - ODL implements several networking technologies allowing to reconfigure each controller's networking services.

Tungsten (Open-Contrail)

Tungsten Fabric (previously known as Juniper's Open-Contrail) is the open-source version of Juniper's Contrail SDN controller, an industry leader for commercial SDN solutions targeting overlay and underlay Networks. Tungsten has two main components: an SDN controller and a virtual Router (vRouter). The SDN controller is composed by three types of nodes:

- Configuration nodes that are responsible for the management layer. They provide a REST API [97] that can be used to configure the system or extract operational status. Multiple nodes of this type can be deployed for HA purposes. Note that configuration states are stored into a Cassandra database (NoSQL).
- Control nodes are in charge of implementing decisions made by the configuration nodes. They receive configuration states from the configuration nodes using IF-MAP protocol and use IBGP to exchange routing information with other control nodes. They are also capable of exchanging routes with gateway nodes using BGP.
- Analytic nodes are responsible for collecting, collating, and presenting analytic information.

The vRouter is a forwarding plane of a distributed router that runs in the hypervisor of a virtualized server. It is responsible for installing forwarding state into the forwarding plane. It exchanges control states such as routes and receives low-level configuration states from control nodes using XMPP.

Although there is not any constraint on how the different nodes should be deployed, Tungsten architecture can be considered as a two-level hierarchical design. Configuration nodes could be seen as root controllers and control nodes as local controllers (hence the configuration nodes can be considered as the leaders). Given the fact that the solution is used by several of the most important actors in industry and that the code can be tested by anyone, a TRL of 9 has been assigned to Tungsten. Tungsten integrates closely with Neutron consuming its API. Since Tungsten supports a large set of networking services, it is configured as a Core plug-in in Neutron.

Addressing the challenges

- Information granularity: Not addressed - although multiple configuration nodes can share the network information through Cassandra, the internal design of Tungsten prevents the deployment of different configuration nodes across different sites. An extension has been proposed to handle multi-region scenarios [98]. However, the extension exposes a centralized entity to orchestrate remote controllers.
- Information scope: Not addressed - the configuration nodes share a global knowledge base. One operation is visible by all configuration nodes.

- Information availability: Undefined - because Tungsten has been designed for a single deployment, the impact of network disconnections between the configuration and control nodes has not been discussed in detail. It is unclear what could happen if a control node cannot interact with the site that hosts the configuration nodes for a long period.

- Automatized interfaces: Not addressed - although control nodes can interact each other, there is no east-west interface to communicate among configuration nodes of different Tungsten deployments.

- Networking technologies: Addressed - Tungsten incorporates several networking technologies and is able to configure different kind of network equipment.

6.3 Summary

In this section, we described twelve solutions that propose to leverage on several controllers to manage virtualized networking infrastructure. Solutions such as FlowBroker, D-SDN, Tungsten and Kandoo use a hierarchy of controllers to gather networking states and maintain a global view of the infrastructure .To avoid the SPOF issue of the root controller (see Section 5.1, most of these systems propose to deploy multiple instances. By deploying as many root controllers as local ones, it is possible to transform such a hierarchical architecture into a distributed one and envision a direct communication between each root controller when needed. The pending issue is related to the global view of the system that needs to be maintained by continuously exchanging messages among the root controllers (i.e., distributed but logically centralized architecture).

To deal with such an issue, solutions such as Elasticon, HyperFlow, Orion, DragonFlow, Onix, and ONOS, use a distributed database, which enables controllers to easily maintain and share global networking information. While it is one more step to fulfill the requirements of the system we are looking for, the efficiency of these systems depends on the capabilities offered by the database system. Even if dedicated systems have been designed for some of them (e.g., ONOS), they do not cope with the requirements we defined in terms of data locality awareness or network partitioning issues.

The two remaining systems, i.e., DISCO and ODL propose a fully distributed architecture (i.e., without the need for a global view). More precisely, DISCO respects the principle of locality awareness and independence of every groups composing the infrastructure: Each controller manages its respective group and peers with another only when traffic needs to be routed to it, thus sharing only service-concerning data and not necessarily global network information. This way of orchestrating network devices is also well fitted in cases of network partitions as an isolated DISCO controller will be capable of providing local domain services. The flaw of DISCO is to provide networking services without the scope of the VIM (i.e., it delivers mainly domain-forwarding operations, which includes only conflict-less exchanges). Offering the VIM expected operations (such as dynamic IP assignment) is prone to conflict and thus might be harder to implement in such an architecture. We discussed this point for ODL, which has a lot of similarities with DISCO (data locality awareness, AMQP to communicate among controllers, etc.). Through the Federation and Netvirt projects, ODL offers premises of a DCI networking service but at a level that does not enable it to solve conflicts. Leveraging the DISCO or ODL architecture and investigating how conflicts can be avoided is a future direction of this survey as underlined in the following section.

7 Directions for DCI networking research

In the previous section, we have studied major decentralized SDN technologies in the DCI context. While we identified that DISCO and ODL are good candidates, there are several open questions and challenges that need investigations. We discuss the most important ones in the following.

7.1 East-West SDN-based interfaces

While the East-West interface has been considered as the principal element to provide inter-controller communications [99], there is still no standard to communicate among controllers and few proposals co-exist [100–102]. This is an issue as such a standard is critical to deliver collaborations between networking modules of multiple VIM instances of a DCI. Moreover, this lack of standardization impacts the communication and automation between North and East-West interfaces. This leads to multiple ad-hoc solutions.

As we outlined in the last section, the East-West interface proposed by DISCO and ODL provides some references to design an efficient horizontal interface for inter-VIMs networking modules communications. Although the analyzed solutions leveraged AMQP as technology to do the East-West interface implementation, other technologies such as REST APIs could be used to provide synchronization and information exchanges among VIMs.

If we consider the model proposed by DISCO and ODL, the use of independent and local databases implies to manage consistency at the application level (i.e., between the different controllers). This entails that the East-West interface should deliver additional calls to resolve conflicts depending on the inter-site service logic used by controllers. Since neither DISCO nor ODL propose a way to manage conflicts at the East-West interface level, this remains as an open question as already highlighted. Another solution could consist in leveraging distributed databases.

7.2 Leveraging new kinds of databases

As we highlighted in the summary of the Section 6, solutions such as Elasticon, HyperFlow, Orion, DragonFlow, Onix, and ONOS, use a distributed database to share global networking information among controllers. While this approach is useful as it is intended to avoid single point of failures and bottlenecks by logically splitting the database, it does not respect the principles of data locality and is not resilient enough in case of networking partitions.

To illustrate this point, one can consider the Cassandra database [82]. Cassandra is based on the principles of a distributed hash table (DHT) to uniformly distribute states across multiple locations based on the computed hash. This means that the states of one specific controller are not necessarily stored in the same geographical location as the controller (e.g., a controller in site A will have states stored in remote sites B, C and so forth). Thus, the principle of locality awareness is not respected as information belonging to a site A will be spread to other sites with no possibility to control the location.

Likewise, a SDN-based cloud architecture that leverages Cassandra will not be resilient to network isolation as it will be impossible for the local controller to retrieve its states, which may have be spread over non reachable sites.

However, data-related approaches different from traditional SQL engines or DHT-based key values systems can be relevant. In particular, solutions such AntidoteDB [103] or Riak [104] that have been designed on top of conflict-free replicated data type (CRDT) [105] could be leveraged by SDN controllers in order to address the aforementioned DCI challenges while respecting the principal characteristics of DCI architectures such as data locality awareness. A CRDT is a

data structure which can be replicated across multiple nodes in a network. Each replica can be updated independently and concurrently. This means that in case of network partitions, each replica will be locally accessible and ready to use. The richness of the CRDT structure is that it is possible to resolve inconsistency between multiple replicas eventually. However, CRDTs come with two important limitations. First, it requires to replicate the state of every site of the DCI infrastructure. Second, only elementary values can be structured as CRDT right now. For instance, it is not sure that a CIDR can be modeled as a CRTD. If solutions for these two problems might be found, CRDT may represent a step forward to provide a distributed solution while respecting the DCI properties.

7.3 Data plane technologies

The eco-system to deliver traffic forwarding between virtual instances is old and extremely rich, with multiple proposals since the initial OpenFlow protocol [26] (BGP [106], SoftRouter [107], RCP [108], as well as RouteFlow [109] to name a few). This eco-system continues to grow with more recent solutions [110–112]. Since heterogeneity in networking protocols may be present in a DCI, the possibility to agree on which mechanisms to use when establishing an inter-site networking service needs to be considered in future works.

7.4 Performance analysis

This survey focuses on defining a general distributed model for DCI networking management based on SDN technologies. While it gives valuable information, it would be relevant to evaluate the selected solutions under the performance perspective. Depending on the analyzed element (e.g., East-West interface, database, or networking protocols) of new proposals, the metrics and analyzed characteristics may vary:

- *East-West interfaces:* The use of a horizontal interface implies that besides the time expended by the system to answer a user request locally (e.g., CPU consumption, memory consumption, threads utilization), the time needed to synchronize with remote sites and provide a final result needs to be taken into account. Such delay will impact the total inter-site service creation time and may be dependent of the technical implementation of the solution (e.g., protocol used). In the architectural design, the quantity of messages, which are needed to establish a service, needs to be optimized to minimize the overhead. Thus, future works should do an analysis comparing the impact on the performance of their inter-site communication model and the technology used to implement it.

- *New kinds of databases:* Similar to the East-West interface, the use of new kinds of database engines will add an extra delay to communicate with remote sites. Moreover, a comparison can be made w.r.t. traditional SQL database replication systems and distributed databases in two different aspects: the local execution time spend by the database to execute CRUD actions (e.g., CPU consumption, memory consumption,...), and the time needed to communicate with remote sites in order to replicate data or synchronize them. Moreover, an important analysis should be done in order to clarify how the database will deal with conflicts or inconsistencies in case of network partitions. Although new database engines provide models based on theoretical assumptions, there are side-effects that can be identified only by conducting experiment driven activities.

- *Networking protocols:* Since the networking route exchange, and traffic forwarding and routing will also impact the time needed for an inter-site service to be effectively deployed,

the performance of the data plane technologies needs to be taken into account. For instance, in the case of BGP VPN routes exchanges, prior works analyzed the benefits and disadvantages of their use in several contexts [113, 114]. Although the implementation of several different networking protocols is a very challenging and complex task, solutions supported by large communities such as OpenStack, ODL or ONOS could promote such kind of development in order to perform further tests and analysis.

- *Consensus protocols application:* If a consensus protocol is mandatory, further investigations will be needed in order to quantify the overall performance of the protocol. Indeed, the round-trip times in a geo-distributed infrastructure could affect the protocol performance by adding latency for which the consensus protocol may not be designed [59]. Moreover, tasks such as leader election may create traffic overload due to the quantity of messages exchanged. For this reason, comparison with static master election or non-consensus protocols will be needed in order to understand the different trade-offs that may be involved.

7.5 Security

Some of the open questions in SDN-based cloud computing solutions rely in the security issues. While security in cloud data centers has been explored in the last decade [115,116], more research in security for SDN and SDN-based solutions is needed [31]. Obviously, the decentralization reduces the impact of having single point of failures into an architecture (e.g., DoS attacks), but some other components need to be revised. For instance, the inter-site communication needs to be secure enough to avoid uncontrolled or non-desired intrusions. The protocols allowing the database distribution may also be deeply studied in order to evaluate the impact of encryption technologies in the overall performance of future solutions. Finally, as a great number of tenants may share the same network medium to access DCI networking services, the isolation and security provided by the networking protocols used will need further studies.

8 Conclusions

Leveraging each Point of Presence (PoPs) in Telcos' architectures as part of a Distributed Cloud Infrastructure (DCI) is an approach that will be mandatory in a near future to address the requirements posed by trends like NFV, IoT, and MEC. Although some resource management systems for this kind of infrastructure have been proposed, providing inter-site networking services in a distributed fashion is still a challenge to be addressed.

This paper explained the problems of centralized networking management in VIMs and how decentralized SDN approaches could represent an opportunity to tackle the inter-site connectivity management challenges. We then presented a survey of several multi-controller SDN solutions and their possible use and limitations as network management tool for a DCI.

Solutions such as FlowBroker, D-SDN, Tungsten or Kandoo propose to maintain a global view of the system through a hierarchy of controllers that does not enable to address the identified challenges.

The use of a distributed database to share global information among the controller proposed by Elasticon, HyperFlow, Orion, DragonFlow, Onix or ONOS, do not entirely address the proposed challenges as the use of the database in a DCI context and under network partitioning is unclear and do not respect the general requirements. We, however, do not eliminate such an approach as new database engines have been recently proposed [103, 117]. We should in particular better understand how these systems behave according to the DCI challenges (data granularity,

scope, and network partitioning issues) while respecting the DCI general requirements (such as the locality awareness).

Solutions following fully distributed architecture, like DISCO and ODL, address several challenges while guaranteeing the general requirements. In particular, we would like to conclude by underlining the potential of a solution like DISCO or ODL to distribute the connectivity management in a scalable, locality-aware, resilient, and easy manner. The work we should address in a short term is to investigate how VIM-related operations can be developed. Our proposal should include the possibility to create on-demand Layer 2 extensions and Layer 3 to span overlay networks among the requested sites at the VIM-level avoiding or solving the possible management conflicts.

A distributed VIM-native tool in which network devices are automatically configured, provisioned, and managed may represent a huge contribution to favor the advent of DCIs such as envisioned in Fog and Edge Computing platforms.

References

[1] Ayoub Bousselmi, Jean Francois Peltier, and Abdelhadi Chari. Towards a Massively Distributed IaaS Operating System: Composition and Evaluation of OpenStack. *IEEE Conference on Standards for Communications and Networking*, 2016.

[2] Adrien Lebre, Jonathan Pastor, Anthony Simonet, and Frédéric Desprez. Revising OpenStack to Operate Fog/Edge Computing Infrastructures. *IEEE International Conference on Cloud Engineering*, 2017.

[3] Deploying Distributed Compute Nodes to Edge Sites. `https://access.redhat.com/documentation/en-us/red_hat_openstack_platform/13/html/deploying_distributed_compute_nodes_to_edge_sites/deploying_distributed_compute_nodes_to_edge_sites`. Accessed: 02/2020.

[4] StarlingX, a complete cloud infrastructure software stack for the edge. `https://www.starlingx.io`. Accessed: 02/2020.

[5] KubeEdge, a Kubernetes Native Edge Computing Framework. `https://kubernetes.io/blog/2019/03/19/kubeedge-k8s-based-edge-intro`. Accessed: 02/2020.

[6] KubeFed, Kubernetes Cluster Federation. `https://github.com/kubernetes-sigs/kubefed`. Accessed: 02/2020.

[7] D. Sabella, A. Vaillant, P. Kuure, U. Rauschenbach, and F. Giust. Mobile-Edge Computing architecture: The role of MEC in the Internet of Things. *IEEE Consumer Electronics Magazine*, 5(4):84–91, Oct 2016.

[8] The OpenStack Foundation. Cloud Edge Computing: Beyond the Data Center (White Paper). `https://www.openstack.org/assets/edge/OpenStack-EdgeWhitepaper\-v3-online.pdf`, Jan 2018. (Accessed: 2020-02-10).

[9] Ronan-Alexandre Cherrueau. A POC of OpenStack Keystone over CockroachDB. `https://beyondtheclouds.github.io/blog/openstack/cockroachdb/2017/12/22/a-poc-of-openstack-keystone-over-cockroachdb.html`, 2017.

[10] Joao Soares, Fetahi Wuhib, Vinay Yadhav, Xin Han, and Robin Joseph. Re-designing Cloud Platforms for Massive Scale using a P2P Architecture. *IEEE 9th International Conference on Cloud Computing Technology and Science*, 2017.

[11] OpenStack. Neutron - Openstack Networking Service. `https://docs.openstack.org/neutron/latest/`, 2020.

[12] Abdelhadi Chari, Thomas Morin, Dina Sol, and Karine Sevilla. Approaches for on-demand multi-VIM infrastructure services interconnection. Technical report, Orange Labs Networks, 2018.

[13] Sriram Subramanian and Sreenivas Voruganti. *Software-Defined Networking (SDN) with OpenStack*. Packt, 2016.

[14] ETSI. Network Functions Virtualisation (NFV) Ecosystem, Report on SDN Usage in NFV Architectural Framework. Technical report, European Telecommunications Standards Institute, 2015.

[15] Marouen Mechtri, Ines Houidi, Wajdi Louati, and Djamal Zeghlache. SDN for Inter Cloud Networking. In *Proceedings of the 2013 IEEE SDN for Future Networks and Services*, pages 1–7, 2013.

[16] Ioan Petri, Mengsong Zou, Ali Reza-Zamani, Javier Diaz-Montes, Omer Rana, and Manish Parashar. Software Defined Networks within a Cloud Federation. In *Proceedings of the 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, pages 179–188, 2015.

[17] Ali Sanhaji, Philippe Niger, Philippe Cadro, Cédric Ollivier, and André-Luc Beylot. Congestion-based API for cloud and WAN resource optimization. *2016 IEEE NetSoft Conference and Workshops*, 2016.

[18] Sushant Jain, Alok Kumar, Subhasree Mandal, Joon Ong, Leon Poutievski, Arjun Singh, Subbaiah Venkata, Jim Wanderer, Junlan Zhou, Min Zhu, Jon Zolla, Urs Hölzle, Stephen Stuart, and Amin Vahdat. B4: Experience with a Globally-deployed Software Defined Wan. In *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, pages 3–14, 2013.

[19] Zhenjie Yang, Yong Cui, Baochun Li, Yadong Liu, and Yi Xu. Software-Defined Wide Area Network (SD-WAN): Architecture, Advances and Opportunities. *28th International Conference on Computer Communication and Networks*, 2019.

[20] ETSI. Network Functions Virtualisation (NFV); Management and Orchestration . Technical report, European Telecommunications Standards Institute, 2014.

[21] Patrick Marsch, Ömer Bulakci, Olav Queseth, and Mauro Boldi. *5G System Design: Architectural and Functional Considerations and Long Term Research*. John Wiley & Sons, 2018.

[22] Michail-Alexandros Kourtis, Michael McGrath, Georgios Gardikis, Georgios Xilouris, Vincenzo Riccobene, Panagiotis Papadimitriou, Eleni Trouva, Francesco Liberati, Marco Trubian, Josep Batallé, Harilaos Koumaras, David Dietrich, Aurora Ramos, Jordi Ferrer, José Bonnet, Antonio Pietrabissa, Alberto Ceselli, and Alessandro Petrini. T-NOVA: An Open-Source MANO Stack for NFV Infrastructures. *IEEE Transactions on Network and Service Management*, 2017.

[23] Thomas Soenen, Steven Van Rossem, Wouter Tavernier, Felipe Vicens, Dario Valocchi, Panos Trakadas, Panos Karkazis, George Xilouris, Philip Eardley, Stavros Kolometsos, Michail-Alexandros Kourtis, Daniel Guija, Shuaib Siddiqui, Peer Hasselmeyer, Josè

Bonnet, and Diego Lopez. Insights from SONATA: Implementing and integrating a microservice-based NFV service platform with a DevOps methodology. *2018 IEEE/IFIP Network Operations and Management Symposium*, 2018.

[24] Siamak Azodolmolky, Philipp Wieder, and Ramin Yahyapour. SDN-based Cloud Computing Networking. *ICTON 2013*, 2013.

[25] L Yang, R Dantu, T Anderson, and R Gopal. Forwarding and Control Element Separation (ForCES) Framework. RFC 3746, RFC Editor, April 2004.

[26] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, JenniferRexford, Scott Shenker, and Jonathan Turner. OpenFlow: Enabling innovation in campus networks. *ACM SIGCOMM Computer Communications Review*, 2008.

[27] Cisco. OpFlex: An Open Policy Protocol White Paper. Technical report, Cisco, 2014.

[28] Bruno Medeiros, Marcos Simplicio Jr., Tereza Melo, Marco Torrez, Fernando Redigolo, Ewerton Rodrigues, and Dino Cristofoleti. *Applying Software-defined Networks to Cloud Computing*. 33rd Brazilian Symposium on Computer Networks and Distributed Systems, 2015.

[29] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, et al. A view of cloud computing. *Communications of the ACM*, 53(4):50–58, 2010.

[30] Siamak Azodolmolky, Philipp Wieder, and Ramin Yahyapour. Cloud Computing Networking: Challenges and Opportunities for Innovations. *IEEE Communications Magazine*, 2013.

[31] JUNGMIN SON and RAJKUMAR BUYYA. A Taxonomy of SDN-enabled Cloud Computing. *ACM Computing Surveys*, 2017.

[32] OpenStack. OpenStack. `https://docs.openstack.org/latest/`, 2020.

[33] Linux Foundation. Kubernetes. `https://kubernetes.io/docs/home/`, 2020.

[34] OpenStack. Neutron Networking-L2GW. `https://docs.openstack.org/networking-l2gw/latest/readme.html`, 2018.

[35] OpenStack. Neutron BGPVPN Interconnection. `https://docs.openstack.org/networking-bgpvpn/latest/`, 2018.

[36] OpenStack. Neutron VPNaaS. `https://docs.openstack.org/neutron-vpnaas/latest/`, 2019.

[37] Docker. Docker. `https://www.docker.com/`, 2020.

[38] Canonical. Linux containers. `https://linuxcontainers.org/`, 2020.

[39] Linux Foundation. Kubernetes Network Plugins. `https://kubernetes.io/docs/concepts/extend-kubernetes/compute-storage-net/network-plugins/`, 2019.

[40] Linux Foundation. Cluster Networking: KubernetesKubernetes. `https://kubernetes.io/docs/concepts/cluster-administration/networking/`, 2019.

[41] Tigera. Calico for Kubernetes. `https://docs.projectcalico.org/v2.0/getting-started/kubernetes/`, 2020.

[42] Cilium. Cilium. `https://cilium.io/`, 2019.

[43] Cisco. Contiv. `https://contiv.io/`, 2019.

[44] CoreOS. Flannel. `https://github.com/coreos/flannel`, 2019.

[45] Weave-Works. Weave Net. `https://www.weave.works/blog/weave-net-kubernetes-integration/`, 2016.

[46] OpenStack. OpenStack kuryr. `https://docs.openstack.org/kuryr-kubernetes/latest/`, 2019.

[47] OpenStack. Tricircle Project. `https://wiki.openstack.org/wiki/Tricircle`, 2018.

[48] Fetia Bannour, Sami Souihi, and Abdelhamid Mellouk. Distributed SDN Control: Survey, Taxonomy and Challenges. *IEEE Communications Surveys & Tutorials*, 2018.

[49] Murat Karakus and Arjan Durresi. A survey: Control plane scalability issues and approaches in Software-Defined Networking (SDN). *Computer Networks 112*, 2016.

[50] Hemin Yang, Jared Ivey, and George F. Riley. Scalability Comparison of SDN Control Plane Architectures Based on Simulations. *International Performance Computing and Communications Conference*, 2017.

[51] Othmane Blial, Mouad Ben Mamoun, and Redouane Benaini. An Overview on SDN Architectures with Multiple Controllers. *Hindawi*, 2016.

[52] Yustus Eko Oktian, SangGon Lee, HoonJae Lee, and JunHuy Lam. Distributed SDN controller system: A survey on design choice. *Computer Networks 121*, 2017.

[53] OSRG. GoBGP. `https://osrg.github.io/gobgp/`, 2018.

[54] The Linux Foundation. OpenvSwitch. `https://www.openvswitch.org/`, 2018.

[55] The Linux Foundation. Linux Bridges. `https://wiki.linuxfoundation.org/networking/bridge`, 2018.

[56] Zhongkui Li, Zhisheng Duan, and Wei Ren. Designing Fully Distributed Consensus Protocols for Linear Multi-agent Systems with Directed Graphs. *IEEE Transactions on Automatic Control 60*, 2014.

[57] Leslie Lamport. The Part-Time Parliament. *ACM Transactions on Computer Systems 16*, 1998.

[58] Diego Ongaro and John Ousterhout. In Search of an Understandable Consensus Algorithm. *USENIX Annual Technical Conference*, 2014.

[59] Ailidani Ailijiang, Aleksey Charapko, and Murat Demirbas. Consensus in the Cloud: Paxos Systems Demystified. *25th International Conference on Computer Communication and Networks*, 2016.

[60] Yang Zhang, Eman Ramadan, Hesham Mekky, and Zhi-Li Zhang. When Raft Meets SDN: How to Elect a Leader and Reach Consensus in an Unruly Network. *Proceedings of the First Asia-Pacific Workshop on Networking*, 2017.

[61] OpenNetworkingFoundation. OpenFlow Switch Specifications. Technical report, The Open Networking Foundation, 2015.

[62] European Commission. Horizon work programme 2020. `https://ec.europa.eu/research/participants/data/ref/h2020/wp/2014_2015/annexes/h2020-wp1415-annex-g-trl_en.pdf`, 2014.

[63] SDXCentral. SDN Controller Comparison Part 1: SDN Controller Vendors (SDN Controller Companies). `https://www.sdxcentral.com/sdn/definitions/sdn-controllers/sdn-controllers-comprehensive-list/`.

[64] SDXCentral. SDN Controller Comparison Part 2: Open Source SDN Controllers. `https://www.sdxcentral.com/sdn/definitions/sdn-controllers/open-source-sdn-controllers/`.

[65] Kevin Phemius, Mathieu Bouet, and Jeremie Leguay. DISCO: Distributed Multi-domain SDN Controllers. *Network Operations and Management Symposium*, 2014.

[66] Mateus Santos, Bruno Nunes, Katia Obraczka, and Thierry Turletti. Decentralizing SDN's Control Plane. *IEEE Conference on Local Computer Networks*, 2014.

[67] Advait Dixit, Fang Hao, Sarit Mukherjee, Lakshman, and Ramana Kompella t. Towards an Elastic Distributed SDN Controller. *HotSDN*, 2013.

[68] Dan Marconett and S Yoo. FlowBroker: A Software-Defined Network Controller Architecture for Multi-Domain Brokering and Reputation. *Journal of Network System Management*, 2015.

[69] Amin Tootoonchian and Yashar Ganjali. HyperFlow: A Distributed Control Plane for OpenFlow. *IEEE Proceedings of the 2010 internet network management conference on Research on enterprise networking*, 2010.

[70] Soheil Yeganeh and Yashar Ganjali. Kandoo: A Framework for Efficient and Scalable Offloading of Control Applications. *HotSDN*, 2012.

[71] Yonghong Fu, Jun Bi, Kai Gao, Ze Chen, Jianping Wu, and Bin Hao. Orion: A Hybrid Hierarchical Control Plane of Software-Defined Networking for Large-Scale Networks. *IEEE 22nd International Conference on Network Protocols*, 2014.

[72] Hazelcast Project. `https://hazelcast.org/`.

[73] Nicira Networks. NOX Network Control Platform. `https://github.com/noxrepo/nox`, 2009.

[74] Jeremy Stribling, Yair Sovran, Irene Zhang, Xavid Pretzer, Jinyang Li, Frans Kaashoek, and Robert Morris. Flexible, Wide-Area Storage for Distributed Systems with WheelFS. *6th USENIX Symposium on Networked Systems Design and Implementation*, 2009.

[75] OpenStack. DragonFlow : Distributed implementation of Neutron within a large DC. `https://wiki.openstack.org/wiki/Dragonflow`, 2015.

[76] Teemu Koponen, Martin Casado, Natasha Gude, Jeremy Stribling, Leon Poutievski, Min Zhu, Rajiv Ramanathan, Yuichiro Iwata, Hiroaki Inoue, Takayuki Hama, and Scott Shenker. Onix: A Distributed Control Platform for Large-scale Production Networks. *OSDI*, 2012.

[77] Pankaj Berde, Matteo Gerola, Jonathan Hart, Yuta Higuchi, Masayoshi Kobayashi, Toshio Koide, Bob Lantz, Brian O'Connor, Pavlin Radoslavov, William Snow, and Guru Parulkar. ONOS: Towards an Open, Distributed SDN OS. *HotSDN '14*, 2014.

[78] Jan Medved, Anton Tkacik, Robert Varga, and Ken Gray. OpenDaylight: Towards a Model-Driven SDN Controller Architecture. *IEEE WoWMoM*, 2014.

[79] Juniper. Contrail Architecture, 2015.

[80] The Linux Foundation. The Open vSwitch Database. `http://docs.openvswitch.org/en/latest/ref/ovsdb.7/`, 2013.

[81] John Ousterhout, John Ousterhout, Ashish Gupta, Ankita Kejriwal, Collin Lee, Behnal Montazeri, Diego Ongaro, Seo Jin Park, Henry Qin, Mendel Rosenblum, Stephen Rumble, Ryan Stutsman, and Stephen Yang. The RAMCloud Storage System. *ACM Transactions on Computer Systems*, 2015.

[82] Apache Software Foundation. Apache Cassandra. `http://cassandra.apache.org/`, 2016.

[83] Cloud Native Computing Foundation. etcd. `http://etcd.io/`, 2016.

[84] OpenStack. DragonFlow : BGP dynamic routing. `https://docs.openstack.org/dragonflow/latest/specs/bgp_dynamic_routing.html`, 2018.

[85] Apache Software Foundation. ZooKeeper: A Distributed Coordination Service for Distributed Applications. `https://zookeeper.apache.org/doc/r3.4.13/zookeeperOver.html`, 2008.

[86] Apache Software Foundation. Apache Karaf. `https://karaf.apache.org/`, 2010.

[87] Open Networking Foundation. Atomix. `https://atomix.io/docs/latest/user-manual/introduction/what-is-atomix/`, 2019.

[88] ONOS. ONOS - Members. `https://onosproject.org/members/`, 2018.

[89] ONOS. SONA architecture ONOS. `https://wiki.onosproject.org/display/ONOS/SONA+Architecture`, 2018.

[90] ONOS. CORD VTN ONOS. `https://wiki.onosproject.org/display/ONOS/CORD+VTN`, 2018.

[91] ONOS. ONOS - OpenStack (Neutron) Integration. `https://groups.google.com/a/onosproject.org/forum/#!forum/onos-discuss`, 2017.

[92] The New Stack. OpenDaylight is One of the Best Controllers for OpenStack. `https://thenewstack.io/opendaylight-is-one-of-the-best-controllers-for-openstack-heres-how-to-implement-it/`, 2015.

[93] OpenDayLight. OpenDaylight SDNi Application. `https://wiki.opendaylight.org/view/ODL-SDNi_App:Main`, 2014.

[94] OpenDayLight. OpenDaylight Federation Application. `https://wiki.opendaylight.org/view/Federation:Main`, 2016.

[95] OpenDayLight. OpenDaylight NetVirt Application. `https://wiki.opendaylight.org/view/NetVirt`, 2016.

[96] OpenDayLight. User stories OpenDaylight. `https://www.opendaylight.org/use-cases-and-users/user-stories`, 2018.

[97] Roy Fielding. *Chapter 5: Representational State Transfer (REST). Architectural Styles and the Design of Network-based Software Architectures(Dissertation).* UNIVERSITY OF CALIFORNIA, 2000.

[98] Juniper. Contrail Global Controller. `https://www.juniper.net/documentation/en_US/contrail3.2/topics/concept/global-controller-vnc.html`, 2016.

[99] E Haleplidis, K Pentikousis, S Denazis, J Hadi-Salam, D Meyer, and O Koufoupavlou. Software-Defined Networking (SDN): Layers and Architecture Terminology. RFC 7426, RFC Editor, January 2015.

[100] Pingping Lin, Jun Bi, and Yangyang Wang. East-West Bridge for SDN Network Peering. *Internet Conference of China*, 2013.

[101] Haisheng Yu, Keqiu Li, Heng Qi, Wenxin Li, and Xiaoyi Tao. Zebra: An East-West Control Framework For SDN Controllers. *International Conference on Parallel Processing*, 2015.

[102] Fouad Benamrane, Mouad Ben Mamoun, and Benaini Redouane. An East-West interface for distributed SDN control plane: Implementation and evaluation. *Computers & Electrical Engineering*, 2016.

[103] INRIA. AntidoteDB. `https://www.antidotedb.eu/`, 2017.

[104] Riak. Riak database. `https://riak.com/`, 2019.

[105] Marc Shapiro, Nuno Preguiça, Carlos Baquero, and Marek Zawirsk. Conflict-Free Replicated Data Types. In *Stabilization, Safety, and Security of Distributed Systems*, volume 6976. Springer, 2011.

[106] Y Rekhter, T Li, and S Hares. A Border Gateway Protocol 4 (BGP-4). RFC 4271, RFC Editor, January 2006.

[107] T Lakshman, T Nandagopal, R Ramjee, K Sabnani, and T Woo. The SoftRouter Architecture. *Proceedings of the ACM SIGCOMM Workshop on Hot Topics in Networking*, 2004.

[108] M Caesar, D Caldwell, N Feamster, J Rexford, A Shaikh, and J van der Merwe. Design and Implementation of a Routing Control Platform. *Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation*, 2005.

[109] M Nascimento, C Rothenberg, M Salvador, C Correa, S de Lucena, and M Magalhaes. Virtual Routers as a Service: The RouteFlow Approach Leveraging Software-Defined Networks. *Proceedings of the 6th International Conference on Future Internet Technologies*, 2011.

[110] Kashyap Thimmaraju, Bhargava Shastry, Tobias Fiebig, Felicitas Hetzelt, Jean-Pierre Seifert, Anja Feldmann, and Stefan Schmid. Taking Control of SDN-based Cloud Systems via the Data Plane. *Proceedings of the Symposium on SDN Research*, 2018.

[111] Adrien Wion, Mathieu Bouet, Luigi Iannone, and Vania Conan. Distributed Function Chaining with AnycastRouting. *Proceedings of the 2019 ACM Symposium on SDN Research*, 2019.

[112] Anduo Wang, Zhijia Chen, Tony Yang†, and Minlan Yu. Enabling Policy Innovation in Interdomain Routing:A Software-Defined Approach. *Symposium on SDN Research 2019*, 2019.

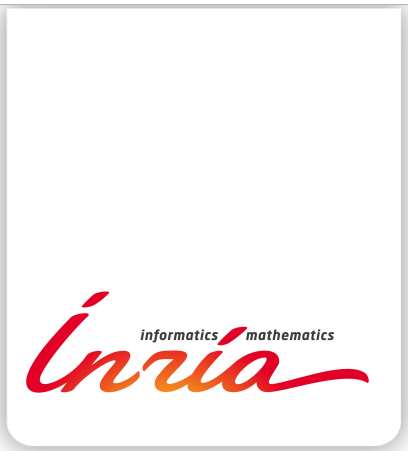
[113] Francesco Palmieri. VPN scalability over high performance backbones evaluating MPLS VPN against traditional approaches. *Proceedings of the Eighth IEEE International Symposium on Computers and Communication*, 2003.

[114] Jian Mai and Jiang Du. BGP performance analysis for large scale VPN. *2013 IEEE Third International Conference on Information Science and Technology*, 2013.

[115] John Rittinghouse and James Ransome. *Cloud Computing Implementation, Management, and Security*. CRC Press, 2009.

[116] Mohamed Almorsy, John Grundy, and Ingo Müller. An Analysis of the Cloud Computing Security Problem. *Proceedings of the APSEC 2010 Cloud Workshop*, 2010.

[117] Karambir Kaur and Monika Sachdeva. Performance evaluation of NewSQL databases. In *2017 International Conference on Inventive Systems and Control (ICISC)*, pages 1–5. IEEE, 2017.



RESEARCH CENTRE
RENNES – BRETAGNE ATLANTIQUE

Campus universitaire de Beaulieu
35042 Rennes Cedex

Publisher
Inria
Domaine de Voluceau - Rocquencourt
BP 105 - 78153 Le Chesnay Cedex
inria.fr

ISSN 0249-6399