Virtual Machine Cluster Mobility in Inter-Cloud Platforms

Stelios Sotiriadis

The Edward Rogers Sr. Department of Electrical and Computer Engineering, University of Toronto, Bahen Centre for Information Technology St. George Campus, 40, Toronto, ON M5S 2E4, Canada

e-mail: s.sotiriadis@utoronto.ca

Nik Bessis

Department of Computing, Edge Hill University, St Helens Rd, Ormskirk Lancashire L39 4QP, UK

e-mail: Nik.Bessis@edgehill.ac.uk

Euripides G.M. Petrakis

Department of Electronic and Computer Engineering, Technical University of Crete Chania, Crete, GR-73100

e-mail: euripides@intelligence.tuc.gr

Cristiana Amza

The Edward Rogers Sr. Department of Electrical and Computer Engineering, University of Toronto, Bahen Centre for Information Technology St. George Campus, 40, Toronto, ON M5S 2E4, Canada

e-mail: amza@ece.utoronto.ca

Catalin Negru

University Politehnica of Bucharest, Romania, Splaiul Independentei 313, Bucharest 060042

e-mail: catalin.negru@cs.pub.ro

Mariana Mocanu

University Politehnica of Bucharest, Romania, Splaiul Independentei 313, Bucharest 060042

e-mail: mariana.mocanu@cs.pub.ro

Preprint submitted to Future Generation Computer Systems *March 8, 2016*

Abstract

Modern cloud computing applications developed from different interoperable services that are interfacing with each other in a loose coupling approach. This work proposes the concept of the Virtual Machine (VM) cluster migration, meaning that services could be migrated to various clouds based on different constraints such as computational resources and better economical offerings. Since cloud services are instantiated as VMs, an application can be seen as a cluster of VMs that integrate its functionality. We focus on the VM cluster migration by exploring a more sophisticated method with regards to VM network configurations. In particular, networks are hard to managed because their internal setup is changed after a migration, and this is related with the configuration parameters during the re-instantiation to the new cloud platform. To address such issue, we introduce a Software Defined Networking (SDN) service that breaks the problem of network configuration into tractable pieces and involves virtual bridges instead of references to static endpoints. The architecture is modular, it is based on the SDN OpenFlow protocol and allows VMs to be paired in cluster groups that communicate with each other independently of the cloud platform that are deployed. The experimental analysis demonstrates migrations of VM clusters and provides a detailed discussion of service performance for different cases.

Keywords: Cloud computing, Cloud portability, Cloud service mobility, Software Defined Architecture, Future Internet Service Migration, VM Cluster Migration

---

1. Introduction

Cloud computing platforms (e.g. OpenStack[1]) offers to Internet users an on demand solution to utilize remote resources. It is comprised by three layers; these are (a) the applications and services, (b) the computing resources (that are virtualized) and (c) the connectivity (including virtual networks). Over the years, various cloud models have been developed to cover the variety of cloud users needs (22), such as the Infrastructure, Platform and Software as a Service (IaaS, PaaS and SaaS). The evolution of these, characterizes the so-called future Internet (FI) concept that allows development of cloud appli-

[1]https://www.openstack.org

cations from services belonging to different owners and developers that is also related with the inter-operation of multiple and different technologies (4). In particular, developers build applications by utilizing on-the-self cloud services encapsulating common functionalities (e.g. user authentication, data storage and context data management etc.), instead of re-engineering and implementing services from scratch.

The FI model has been characterized particularly useful for a large community of developers and Internet entrepreneurs for Internet-enabled innovation specialized in cloud solutions that use enablers. These are cloud services that provide open specifications, based on a cloud-centric service environment as in (1) and are available for utilization through APIs (e.g. RESTFul based). While these solutions highlight innovation and promotion of a new easy-going development method, software engineering processes are becoming more and more complex. This is because such enablers have distinct features that impact several different research fields as follows.

- Executed over virtualized hardware and other software stacks that can be highly scalable according to the needs of the users (e.g. elasticity in terms of resources). For instance the users can increase their computational capacities according to the users' demand.

- Share common physical resources with other cloud applications or services, thus supporting multi-tenancy environments. For instance an enabler could be utilized by more than one cloud applications, thus it requires to have high quality of service.

- Provide interfaces using flexible and easy to use APIs that allow combination of services, remote utilization and management.

- Utilize third party services that are decentralized and involve a multi-layer structure e.g. belonging to different cloud platforms or owned by different providers. For instance, a service could be integrated upon another service that is already developed by a collection of other services and so on.

- Are interoperable but could be based on different semantics and information models with regards to communication schemas (JSON, XML etc.) or communication protocols.

In this study we focus on the connectivity layer of applications and services in clouds and inter-clouds with the aim to explore the way in which software defined networks (SDN) could assist in the direction of efficient VM migration among resources for communicating applications. This involves multiple resources from different cloud providers so to create distributed virtual computing clusters as in (7) and (27). In a cloud system, Virtual Machines (VMs) are configured by the cloud platform to operate under the datacenter setup, thus making portability to other systems a really challenging issue. In particular, cloud networking is one of the aspects that highlights important issues in VMs migration due to the internal network configuration. For example a running VM includes static MAC addresses, floating IPs and other setup that is assigned by the cloud platform. In case of a VM migration a new configuration will happen and existing setup will be lost. The case becomes worst when a cluster of VMs are migrated, thus reconfiguration causes VM links to be lost.

Having said that, in this work we focus on the portability of VM clusters that are multiple VMs communicating with each other, and we address the following challenges.

- How to easily configure VM cloud networks without worrying for their static configurations? This is because communication setup in VMs is realized by means of static IPs.
- How to dynamically change network configurations dynamically?

To answer these challenges, we propose an SDN architecture that is cost-effective, manageable, agile and allows re-configuration of VMs on the fly by decoupling the cloud platform network setup from the VMs. The proposed solution is based on OpenFlow[2] that is an open standards-based and vendor-neutral protocol. SDN targets specially on the separation of the (a) control plane of the network that is responsible for decisions making with regards to how packets travel through the network (could be happened remotely) from (b) the data plane of the network that is responsible for moving packets from one place to another and is local based. OpenFlow provides a remote controller with the capability to modify the behavior of network devices, through a well-defined "forwarding instruction set" [3]. Typically, cloud

[2]http://archive.openflow.org

[3]http://archive.openflow.org/wp/learnmore/

platforms (e.g. OpenStack) have OpenFlow support with regards to their internal virtual networks.

A technical problem is that VM endpoints are static and refer to the floating IP addresses assigned by the cloud itself and used by the cloud application developers in various parts within their source code. So, in case of a VM migration the developers will need to adapt internal code sources to match the configurations of the new hosting cloud platform. The SDN architecture allows cloud services to be flexible and dynamic with regards to their network configurations that could be in turn modified in a vigorous way during system run-time. The solution couples VMs as communities by utilizing internal bridges that are paired to their static IPs and could be changed on demand in case of a VM migration (thus using the SDN control plane).

According to this discussion this work contribution is on the VM cluster migration using an SDN architecture and includes the following.

(C.I) We separate the VM internal network setup from the cloud platform network configuration that is assigned by the cloud platform it self. By this way, VMs could be easily migrated to different clouds overcoming the issues provoked by static setup (e.g. IPs, MAC addresses etc.).

(C.II) We design an adaptive architecture for redirection of traffic among VMs when migration occurs. The SDN architecture allows adaptations of the VM traffic that is automatically configured by a VM cluster migration module. This is responsible for collecting network configurations (from the new cloud platform) and to redirect the VM traffic according to the new endpoints. We provide an experimental analysis of VM migrations in OpenStack cloud platforms to evaluate our solution.

In Section 2 we present the research problem and the motivation of this work, in Section 3 the analysis of the related approaches and the significance of our work, in Section 4 we detail the conceptual model for inter-cloud VM migration and in Section 5 the SDN architecture for VM migration in clouds and inter-clouds by focusing on the OpenStack platform. Then, in Section 6 we demonstrate the performance analysis of OpenStack related use cases and the evaluation of the architectural concepts. Finally, in Section 7 we conclude our study with the key findings and the future research steps.

2. Problem description and Motivation

This section specifies the research problem of this work by demonstrating a use case example. A service called *x service* is deployed in an OpenStack platform, so it has a network configuration according to the platform network service (e.g. Neutron or Quantum based on the OpenStack version) including an endpoint address (characterized by two IPs that are the local and floating IPs). The *x service* provides an API that is accessible by its floating IP address so other applications can access it using the HTTP protocol (for example to fetch data). Let us assume that a new developed application utilizes the *x service* and within the programming code the developer includes the endpoint of the service in the form *http://IP/API*. In a non-migration environment the traffic is managed by the cloud platform and the endpoint remains static during the service lifecycle.

In case of switching to a VM cluster migration environment, meaning that (a) the application it self could move to different cloud location or (b) the services that integrate it could move to different location, it changes affect directly the operation of the service. In particular, the developer should adjust code sources manually and possible restart services and servers to apply changes, an action that affects downtimes and QoS either from the application or the service point of view. This also involves violation of the service level agreements (SLAs) between (a) user and application and (b) application and service. To solve this issue we introduce an SDN architecture that is flexible and agile and allows elimination of public endpoints in terms of static IPs and internal MAC addresses.

This work is motivated by the interoperability and portability scenarios of the 2014 Cloud Standards Customer Council[4] that includes the following points with regards to cloud users.

- They could switch between providers on demand.
- They could use services from multiple providers to integrate a new one.
- They could link cloud services in a multilayer format.
- They could link in-house capabilities to cloud services.

[4]http://www.cloud-council.org/CSCC-Cloud-Interoperability-and-Portability.pdf

We are also motivated by the opportunities of the FI concept that allows development of innovative applications from services belonging to different owners and developers called as cloud enablers. In particular, developers build applications by utilizing on-the-self services offered as cloud enablers encapsulating common functionalities (e.g. user authentication, data storage, context data management etc.), instead of re-engineering and implementing services from scratch. These are usually deployed in cloud systems, provide open specification and are available for utilization through APIs (e.g. RESTFul based). While these solutions highlight innovation and promotion of a new easy-going development method, software engineering processes are becoming more and more complex.

This work focuses on the management of cloud service cluster migration, and we introduce the notion of "live portability" in large scale cloud systems. The assumption is that VMs have local logical network topologies and communicate using static endpoints. We are thus focus on the need to define a framework to hide the abstraction of network using a service oriented network subscription service. The SDN architecture allows adaptations in the VM traffic on request that is automatically configured by a VM cluster migration module that is responsible for collecting network configurations (from the new cloud platform) and to redirect the VM traffic according to the new endpoints.

3. Review of related approaches

This section presents the related works with regards to cloud migration and on the usage of software defined networks in cloud systems to address related issues.

In (15), authors presented a "live ensemble migration solution"that operates as a network operators management tool. The work utilizes recent advances in SDN, with "LIME" a solution that supports network migration of VMs. The authors demonstrate how to migrate an entire network in a way that "both supports arbitrary controller application software and efficiently orchestrates the migration process" (15). The work of (11) is based on "LIME" for network migration of VMs. It is designed to support transparent migration of VMs and switches as well as a technique that clones VMs in order to minimize performance disruptions. This allows the transformation of multiple physical switches to a single virtual that does not affect traffic

and rules updating during the migration. This work expands the work of (15) by including an improved algorithm a discussion on the evaluation part.

The work of (2) includes the VMFlock that is a migration service that allows cross-datacenter transfer of a group of VMs and their instantiation based on their images. The solution utilizes data deduplication that is a specialized technique for data compression to eliminate duplicate copies of repeating data. This is executed "within the VMFlock to be migrated and between the VMFlock and the data already present at the destination datacenter" (2). It also supports data transfer prioritization and acceleration of VMs and application startup. The authors claim that it provides an incrementally scalable and high performance migration service. (11) authors suggest that this work does not include virtual network support and the benefits derives from this approach.

In (6) authors present an approach that allows linked VMs to be co-located within a cluster system on the same physical host to reduce communication costs. Their solution is based on live gang migration, and to reduce the overhead of concurrently migrating multiple co-located VMs they utilize data deduplication. Similarly to (2) the work does not consider virtual network benefits during migration.

In (10), authors present a study to highlight that cannot be a one-to-one mapping between virtual abstractions and their actual physical implementations. They also present research directions that include (a) development of more advanced mapping techniques and (b) to restrict the API of SDN so to provide safe-to-map abstractions only. In (30), authors present an experimental live migration performance study that focuses on the overheads of virtual clusters and explore different virtual clusters migration strategies. The describe a framework called VC-Migration (virtual cluster migration) to manage the migration process. Similarly to (6; 2) this work does focus on the SDN and its benefits (11). Further, in (20) authors consider a mechanism that allows migration of an entire virtual network from an initial to the final mapping. They present "scheduling algorithms for virtual network migrations that minimize the disruption to the current data traffic and the overhead traffic in the migration process"(20) . However, according to (11) the nature of this work is different since the solution aims to change the topology to direct traffic to a new location.

In (16), authors present router grafting, "where parts of a router are seamlessly removed from one router and merged into another by allowing grafting a border gateway protocol (BGP) session and the underlying link

from one router to another, or between blades in a cluster-based router"(16). Similar to (20), and according to (11) the solution aims to change the topology to direct traffic to a new location. In (8), authors propose XenFlow that is a hybrid virtualization system utilizes Xen and OpenFlow technologies. XenFlow goals are to provide (a) a flexible virtual network migration, (b) a strong isolation of virtual networks to avoid deny of service caused by interference of other virtual networks and (c) to offer inter-network and intra-network QoS provisioning by a consistent resource controller (8). In (17), authors present the design and implementation of a network virtualization solution for multi-tenant datacenters. They describe the design and implementation of a network virtualization platform, that has been deployed in production environments. Authors of (11) suggest that these approaches work for software switches and routers exclusively.

In (23), authors "take a step toward the goal of generalizing elasticity by observing that a broadly deployed class of software is particularly well suited to dynamic scale" so as networks become increasingly virtualized, FreeFlow addresses a need for elasticity in middleboxes, without introducing the configuration complexity of running a cluster of independent middleboxes (23). In (9), authors present a work for migrating VMs along with their connections. Authors suggest that this has numerous benefits in data centers, including improved load balancing, energy consumption optimization (power savings) as well as optimization of performance and utilization. However, directly migrating individual components can lead to inconsistencies and overloads of resources (9). In their work they show that by following "an ordering over changes to the virtual network, we can perform this migration efficiently while providing strong guarantees on the ability to meet constraints on bandwidth and loop-freedom" (9). The work of (11) suggests that previous works require application source-code modifications that rely on some topological assumptions.

In (5) authors present Remus that is a solution for extremely high degree of fault tolerance. The authors claim that "a running system can transparently continue execution on an alternate physical host in the face of failure with only seconds of downtime, while completely preserving host state such as active network connections" (5). The solution supports cloning of VMs but not virtual networks. In (12) authors present SWAN that enables an efficient and flexible inter-DC (datacenter) WAN by coordinating the sending rates of services and centrally configuring the network data plane by focusing on updates of a network policy without causing transient congestion (12). In

(13) authors present B4 that is a private WAN connecting Google data centers across the planet. This has a number of unique characteristics such as massive bandwidth requirements, elastic traffic demand and full control over the edge servers and network.

In (3) authors present the design, implementation, and evaluation of a system for supporting the transparent, live wide-area migration of virtual machines that use local storage for their persistent state. in (19) authors presents zUpdate, to perform congestion-free network updates under asynchronous switch and traffic matrix changes. The work formulates "the update problem using a network model and apply our model to a variety of representative update scenarios in DCNs" (19). These works, similar to (12; 9) present ways to update a network policy without causing transient congestion.

In (18), authors present SnowFlock that uses VM fork. The work enables "the straightforward creation and efficient deployment of many tasks demanding swift instantiation of stateful workers in a cloud environment, e.g. excess load handling, opportunistic job placement, or parallel computing" (18). In (24) authors present a system for virtual machine live migration that includes disk images. The approach proposes using copy-on-write images to derive users' custom images from a root image. We are driven by the works in (26), (28) where authors present an inter-cloud job scheduling framework that implements policies to optimize performance of participating clouds. The framework, named as Inter-Cloud Meta-Scheduling (ICMS), is based on a novel message exchange mechanism to allow optimization of job scheduling metrics. The resulting system offers improved flexibility, robustness and decentralization. In (25) we presented an inter-cloud bridge system that is elastic, easy to be upgraded and managed. This work is based on this prototype that is composed not only from heterogeneous cloud platforms but also from independent cloud services. Finally, we detailed an experimental analysis to show interactions with various heterogeneous cloud platforms.

Different from aforementioned solutions this work focuses on the problem of transparent VM migration among different clouds, where live migration is not available and by highlighting the issue of heterogeneity and seamless integration of VM clusters without changing their network configurations. We are motivated by the (14) and (11) that points out the SDN benefits over traditional solutions including:

(i) Effective management of the network layer with regards to the inflexi-

bility and complexity of the traditional network. The proposed solution will provide advances in programmable network interfaces.

(ii) Virtualization provokes cloud datacenter scalability issues, especially when more and more VMs are created and migration is a key requirement for better cluster management. We focus on providing automatized VM group migration among clouds without disrupting communication among VMs.

(iii) Utilization of dynamic networks that can be configured and become adaptable according to changing needs. This includes re-configuration on-the-fly according to the cloud platform setup.

We further differentiate this work from the internal cluster migration including process/memory and suspend/resume migration as the works presented in (21). We aim to the problem of inter-cloud VM cluster migration and on the way that VM clusters could be moved from one place to another and re-instantiated without loosing their connections. To achieve this we (a) separate the VMs internal network configurations from the cloud platform network configuration that is assigned by the cloud platform and (b) we design an SDN architecture allows adaptations in the VM traffic on request that is automatically configured by a VM cluster migration module that is responsible for collecting network configurations (from the new cloud platform) and to redirect the VM traffic according to the new endpoints.

4. Conceptualization of the live portability in cloud computing systems

This section presents the conceptual solution of the live portability that is defined as follows.

Live portability is the process of moving cloud entities, that represent cloud applications and/or services, are composed by different sub-entities (3rd cloud party services and tools), are configured to integrate as a whole system and belonging to same or different physical spaces, from one location to another based on certain properties such as SLAs and QoS constraints.

To demonstrate a live portability system we present an example case. In particular an application called *App* is deployed in Cloud_B and is composed by three services namely as S_{A1}, S_{A2} and S_{C1} that is a service belonging to a different cloud provider (Cloud_C). The assumption is that we need a live migration process when the application owner realizes that a new cloud

provider (Cloud_B is now available and could host the *App* by offering better terms (e.g. better pricing model and improved QoS or in case of a need of more resources) and also provides the same service S_{A2} at improved cost. Let us assume that the owner decides to move *App* as well as S_{A1} (that it will become S_{B1} from cloud to cloud.

To achieve this inter-cloud mobility we developed the inter-cloud bridge system introduced in (26) and implemented in (25) that encompasses a collection of services aiming to ensure modularity and management of key components of an interoperable large scale cloud system composed by heterogeneous clouds. The services designed on a modular basis of interacting RESTFul compliant cloud services. Each module is implemented as a separate service that could be executed in a decentralized topology thus modules could by easily replaced and/or updated with flexible configuration according to the needs of the inter-cloud administrator.

Initially, a user can access the inter-cloud registry using the Identity Management Service and configures the mediation service using the registry. Then, the Mediation Service connects to the configured clouds and collects data regarding generated authentication tokens, images and snapshot, flavors and resource usage information. The collected information is forwarded to the interacted modules (Context Broker, CEP etc.). Also, the External Third Party Cloud Service includes monitoring of applications composed from services belonging to the inter-cloud bridge system. Further, it provides an endpoint for configuring an inter-cloud market place for subscriptions to inter-cloud offerings. More details are presented in (25). In this work we introduce the SDN module that allows the extra feature of dynamic network configurations and described in the next section.

5. Software Defined Architecture for VM Cluster Migration

This section presents the SDN VM cluster migration architecture for moving a cluster of coupled VMs to different cloud platforms. We utilize an SDN networks solution to isolate the communication between VMs participating in the cluster so to achieve nested mobility of VMs in a transparent way among heterogeneous cloud platforms. The architecture is flexible and includes an internal VM configuration based on two physical networks as follows:

(i) The "data network" for VM data traffic based on virtual local area networks among VMs (e.g. VLAN among ethernet interfaces). The assumption here is that the physical switches support such functionality.

(ii) The "management network" for associating VM hosts to static endpoints that are their URIs assigned by the cloud platforms (e.g. floating IP addresses for remote access). The solution will allow updates of VM endpoints on-the-fly according to the cloud platform setup.

Having said that, next we present (a) the VM cluster migration solution that expands the inter-cloud bridge system (Section 5.1, (b) the extensibility of this solution in OpenStack, (Section 5.2) and (c) the VM migration between two OpenStack systems (Section 5.3).

5.1. VM Cluster Migration Solution

This section presents the SDN architecture that develops the internal communication links among VMs. Figure 1 demonstrates the basic solution of the SDN architecture for VM cluster migration.

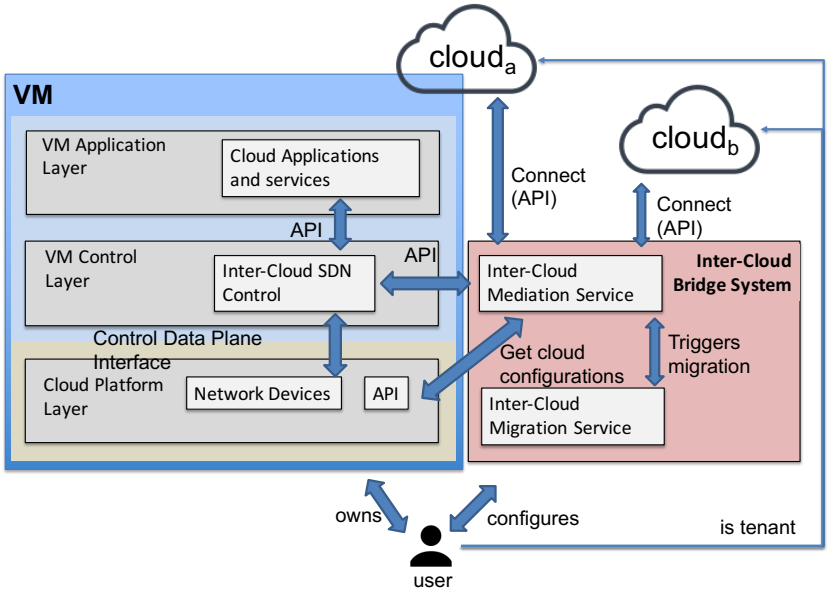

Figure 1: The solution of the SDN architecture for VM cluster migration

The user is a tenant in both clouds and owns a VM, for example a VM in cloud$_a$) and configures the inter-cloud bridge system including authentication credentials as well as it triggers migrations using the inter-cloud migration service. The architecture separates the SDN solution that includes the VM application, control and cloud platform layers and the inter-cloud migration

solution that includes the inter-cloud bridge system that encloses the mediation service. Each module is implemented as a separate service under a different endpoint. The SDN architecture includes the following new components.

(1) The inter-cloud bridge system that is extended to include the following.
   (a) The mediation service that connects to the cloud platforms using their APIs (e.g. OpenStack API). The service collects information about VMs (using HTTP calls) from internal services and it triggers actions e.g. create new VMs as well as configurations (assigning security rules and new floating IP to cloud instances).
   (b) The migration service that includes the process of moving a running instance (and its initial configuration e.g. pre-installed software) from a cloud to cloud while maintaining its hardware, software and network configurations as in (29). After migration the private cloud includes a ready running instance in respect to setup, rules, security group and a public IP.
(2) The application layer defines the static internal endpoints for the cloud applications and services that communicate with the cloud control layer using an SDN configuration. These represent the URI references that could be used by the developers within their source code.
(3) The control layer includes the inter-cloud SDN control module that is dynamically and automatically configured by the inter-cloud mediation service, on-demand and on-the-fly, performing static endpoint updates. Each time a new VM is generated to a different cloud platform, the mediation service automatically updates the endpoints by making a call to this service.
(4) The cloud platform layer defines the various network devices such as switches and routers that communicate with the inter-cloud SDN controller using the OpenFlow protocol. The layer includes the virtual network abstraction of the cloud platform it self.

Figure 2 demonstrates the flow of the HTTP traffic of a VM that follows the proposed SDN architecture including structuring the SDN application and control layer (as presented in Section 1) and includes the following.

(1) The virtual interface that is the internal URI (static endpoint for developers).

(2) The encapsulated interface that is the external URI (dynamic endpoint assigned by the cloud platforms).
(3) The inter-cloud bridge and bond levels and the VM physical interfaces.

The assumption is that multiple VMs follow the same configuration and are paired in cluster groups, that are identified by their inter-cloud bridge (*icbr*) interface. Also, the applications/services refer to internal URIs while the external URIs defined by the cloud platforms are dynamically re-configured on change.

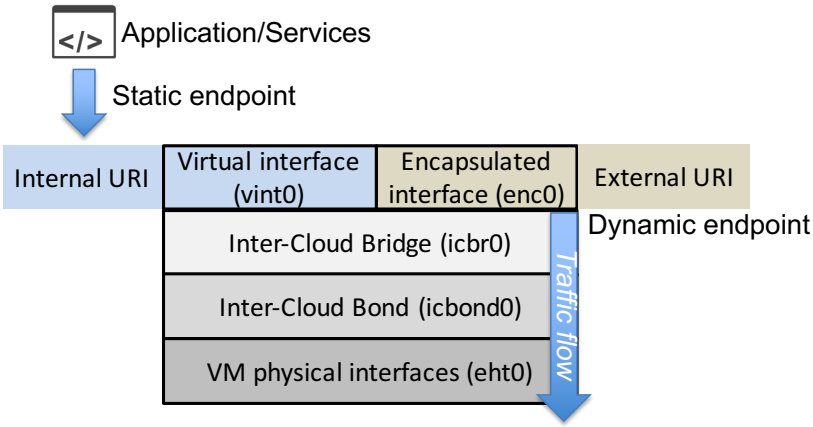

Figure 2: The flow of the HTTP traffic in the internal VM interfaces according to the proposed SDN architecture

We further detail the levels of interfaces for Figure 2 as follows.

(i) Internal URI: The *vint0* is the internal interface of the VM that defines the local endpoint IP of the service. Other applications and/or services that require communicating with the VM will direct traffic to this interface. This also represent the internal static endpoint of the VM that is independent of the cloud platform setup and configured by the user (in the inter-cloud SDN Control module).
(ii) External URI: The *enc0* is the encapsulation inteface that defines the dynamic URI of the VM it self. In case of of a VM mobility to a different cloud platform this is the endpoint required to be changed according to the new assignment. The prototype demonstrates how encapsulated interfaces could be changed dynamically according to the floating IPs of the new cloud platforms. We include the following network virtualization methods:

(i) Virtual extensible LAN (VXLAN) that utilizes UDP so all routers properly distribute traffic by using a hashing over the 5 tuple that include the UDP source and destination ports.

(ii) Generic routing encapsulation (GRE) tunnelling for point-to-point links over the Internet for cases where IP packets do not contain the information necessary to construct a 5-tuple.

(iii) Inter-cloud bridge: The *icbr0*is the interface to create a virtual bridge among the inter-cloud VM networks. Using this interface the VM cluster members redirect their traffic in the inter-cloud system.

(iv) Inter-cloud bond: The *icbond0* is the network bonding that exists within the same cloud platform for optimized throughput and redundancy. The bond is utilized in cases of failures (e.g. active passive when a NIC goes down), aggregated VMs (e.g. multiple NICs act as one to increase throughput) and load balancing cases (for optimized traffic distribution among VMs).

(v) VM interfaces: The *eth0* etc. are the physical interfaces of the VM for traffic redirection to the Internet channel. The VM traffic is forwarded to the interfaces from the inter-cloud bridge to the Internet.

5.2. OpenStack VM Cluster Migration Solution

In this section we design an OpenStack platform VM cluster migration based on the architecture presented in Section 5.1. Figure 3 demonstrates an OpenStack VXLAN setup according to a typical OpenStack topology[5]. Each VM includes the layered structure presented in previous section and includes (a) the static endpoint(s) (in the *vi0* interface) that are the reference URIs used by application/service developers and (b) the dynamic endpoints (*enc0*) that change according to the inter-cloud bridge system. The traffic is directed from internal network (*vi0*) to encapsulated network (*enc0*) that in turn forwards it to the VM physical interface (*eth0*) using the inter-cloud bridge (*icbr0*).

Then, it is handled by the OpenStack bridge (*brint*) and throughout various interfaces (*intbr-eth1* and *phybr-eth1*) is redirected to the node physical host interface (*eth1*) that communicated with the network node. The latter, uses the *eth1* interface to redirect internal traffic to compute nodes (e.g. Compute node 1) and *eth0* for internet connection. It should be mentioned,

[5]http://docs.openstack.org

that the network node is configured to use two virtual bridges the *br-ex* and *br-eth1*. The network tap provides a way to access the data flowing across the inter-cloud network. Also, it includes an external gateway (*eg*) that is a portal between virtual OpenStack network and the internet, and a virtual router that provides a routing framework so the host machine to act as a typical hardware router over a virtual local area network.

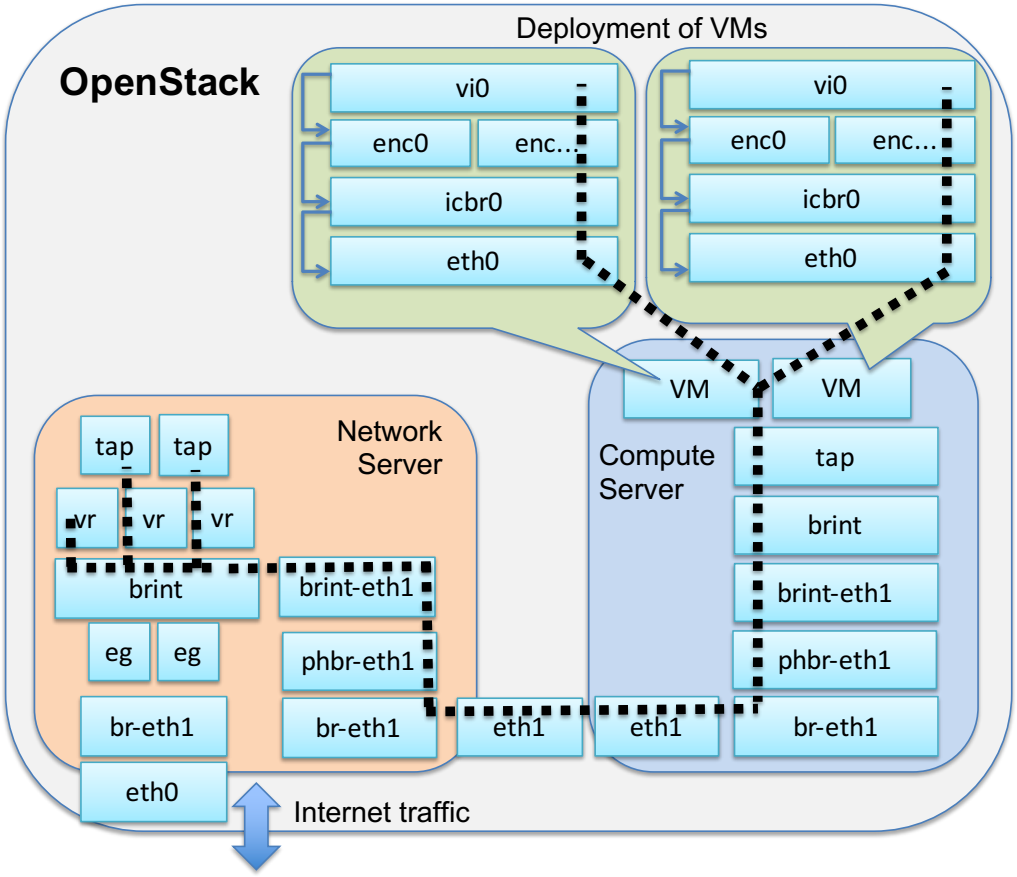

Figure 3: Analysis of the OpenStack VXLAN setup

5.3. VM Mobility Case

To describe the mobility case, we present an example where a VM is migrated among two cloud platforms. The inter-cloud bridge system is responsible for the migration process, while the SDN inter-cloud controller dynamically changes the external endpoints. Figure 4 demonstrates a mobility

of a VM where the URI_a is the static and URI_b is the dynamic endpoint. The assumption is that the service is utilized by an external third party service that has been pre-configured to communicate using the SDN solution.

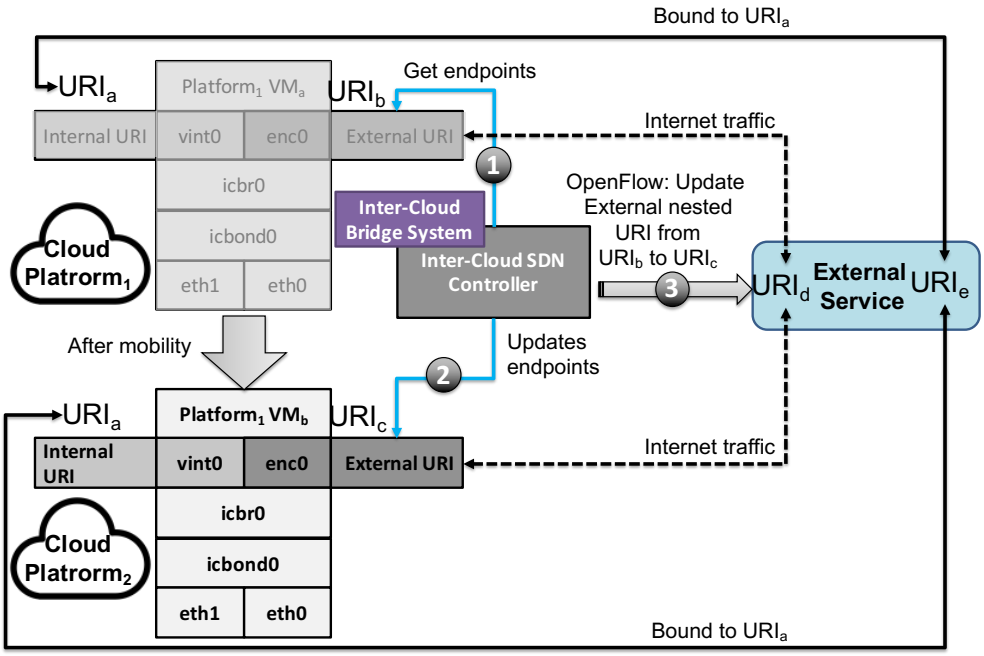

Figure 4: A mobility of a VM where the URI_a is the static and URI_b is the dynamic endpoint data

In case of the inter-cloud mobility from $Platform_1VM_a$ to $Platform_2VM_a$ the service is re-instantiated and the new floating IP assigned by the cloud is configured automatically to the new instance. Thus is this case the following actions are taking place.

(1) The inter-cloud SDN controller collects the URI_b endpoint from the $Platform_1VM_a$. The inter-cloud bridge system operates on top of the controller and triggers updates on change.

(2) Then the controller, by using the OpenFlow protocol, it updates the endpoint list (in this case URI_b it becomes URI_c). It should be mentioned that endpoints URI_a remains the same before and after migration.

(3) The controller updates the external service URI endpoint (URI_d) and the traffic flows as normal from static URI_e to dynamic URI_d, and then to the migrated VM and to the dynamic URI_b to URI_a.

It should be mentioned that during the update process a delay and/or failure in the communication could be observed. The experimental section presents a detailed discussion of such issues and possible configurations to minimize service downtimes.

6. Performance Evaluation

This sections presents the performance evaluation of various mobility cases of the SDN architecture. We detail a number of experimental cases to explore benchmarks as follows.

(I) The inter-cloud mobility times for VM migration among cloud platforms. For this experiment we utilize an OpenStack inter-cloud to demonstrate the various steps of migration and the total times required to create a new running instance (that will be a clone of the original VM). We also migrate different sizes of VM images in order to explore how VM size affects mobility times.

(II) The SDN controller configuration times with regards to the following.
   (a) The time to create the SDN bond among VMs.
   (b) The time to update the SDN controller (when a migration has occurred).
   (c) The downtime during the update process.

(III) The migration of a VM cluster (4 VMs) within the inter-cloud system by measuring features as previously.

6.1. Inter-Cloud Mobility Benchmarks

This experiment presents the inter-cloud mobility benchmarks with regards to the time for a VM to be migrated between different systems as in (29). To achieve this, we separate inter-cloud mobility into a number of processes including the following.

(i) Source cloud authentication: The inter-cloud bridge system connects to the source cloud platform (the cloud where the running VM resides) in order to get security token on demand using the cloud tenant information (username, password and tenant id). The process connects automatically to the source platform and uses the tenant token to collect internal cloud information (running VM instances, images etc.).

(ii) Image creation: The system creates a new image of the selected running instance (VM) that the user requires to move to the target cloud. It should be mentioned that this process allows multiple VMs images creation at the same time.

(iii) Image download: This is the process to download the VM images locally to the inter-cloud bridge system and to assign a unique identifier to the image. The process is executed automatically.

(iv) Target cloud authentication: This allows connectivity with any target clouds (the clouds where the images need to be loaded) that is part of the inter-cloud bridge system and authentication using the tenant information.

(v) Image upload: The process to upload the image(s) from local space to the desired target cloud platform.

(vi) Security setup: The process to configure any security parameters (e.g. public keys that could be exported and imported among cloud platforms) that are associated with the images.

(vii) Instantiation: The process of creating the new instances in the target cloud platforms based on the cloud platform setup.

(viii) Network setup: The inter-cloud bridge system updates the network configurations according to the target cloud setup. The new endpoints are collected by the inter-cloud bridge systems on demand and information is shared with the inter-cloud SDN controller. In turn, this module updates the VM virtual networks and performs a sanity check to ensure connectivity. In case of a multiple VM migration the module updates the VM cluster migration solution following the model of Figure 1. The system makes adaptions to the external URIs that is the *enc0* encapsulation interface that defines the dynamic URI of the VM it self.

Figure 5 demonstrates the different times in milliseconds (ms) for each process of the service to execute a VM migration. We present two cases, migrating a VM of (a) 800 MB size and (b) 1060.94 MB. Based on this figure we can define the benchmark measurements of this study. The most time is spent during image download (40 to 50 seconds) and upload (around 10 seconds) phases, a process that is directly related with bandwidth speed. In addition, the instantiation phase is also time consuming as it requires an average of 8 seconds. The security setup and network configurations are low time consuming actions (less than 4 seconds). Finally, cloud authentication

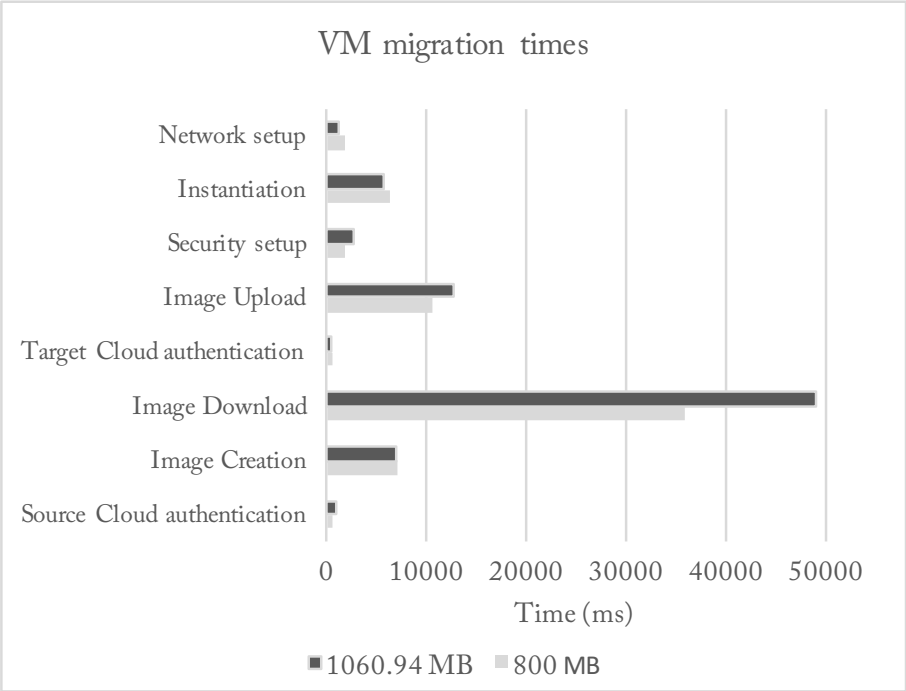

Figure 5: VM Migration times

is related with the time required for an HTTP get request. It should be mentioned that the total migration time (from source to target cloud) of the 800 MB VM image is 64.92 seconds and of the 1060.94 is 79.97 seconds.

6.2. SDN Controller Configuration Times

This section presents the performance of the SDN controller with regards to (a) the time to create the SDN bond among VMs, (b) the time to update the SDN controller (when a migration has occurred) and (c) the downtime during the update process.

6.2.1. Linking time

We define as linking time the time period requiring to create the internal network VM configurations as presented in Figure 2 in both VMs. We present the linking times of 2 VMs in Figure 6. We run 10 pairing experiments, where each of which creates the internal and external SDN networks and we measure the time required for the VMs to configure the *vint0, enc0, icbr0, icbond0 and the eth0*interfaces. It is shown that the average linking time is

63 milliseconds, a time that compared to the migration values of Figure 5 is significantly low.

6.2.2. Update time

We define as update time the time period requiring to update the network configurations when a VM migration has been occurred. This includes update of the encapsulated interface with regards to the external endpoint that is the new floating IP allocated by the OpenStack system. The experimental analysis shows that the average update process is 26.5 ms, a value that if we compare it with the times required to create the network configurations of a new VM (as shown in network setup measurements of Figure 5 is significantly low. Figure 6 demonstrates the Update times for migration of two VMs.

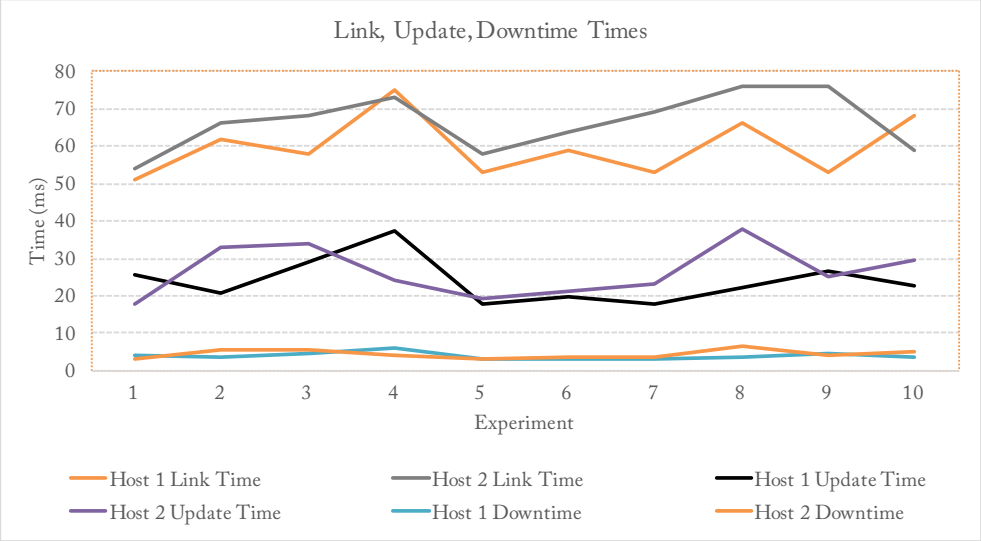

Figure 6: VMs linking times of two VMs

6.2.3. Downtime

We define as downtime the time period during of the update time that the communication between the VM cluster is temporarily broken due to the endpoint update process. In particular, each time the inter-cloud mediation service updates the encapsulated interface, the SDN controller recreates a new interface and it assigns to it the new endpoint. The average downtime is measure as 5.3 milliseconds, that could be considered as a tiny timeframe.

In cases of a whole new VM cluster migration the downtime is insignificant, however in cases of a VM migration where one or more VMs of the cluster are migrated and others stay online this will provoke a downtime to the online VMs. Figure 6 demonstrates the downtimes for migration of two VMs.

6.3. VM Cluster Migration

This section includes the migration and configuration of a centralized four VMs cluster in an OpenStack system. The cluster is centralized as all VMs reside at the same cloud platform and have similar functionalities (e.g. migration of a database cluster). Figure 7 demonstrates a use case of VM cluster migration (of 3000 MB image size) and the various configuration steps including the following data.

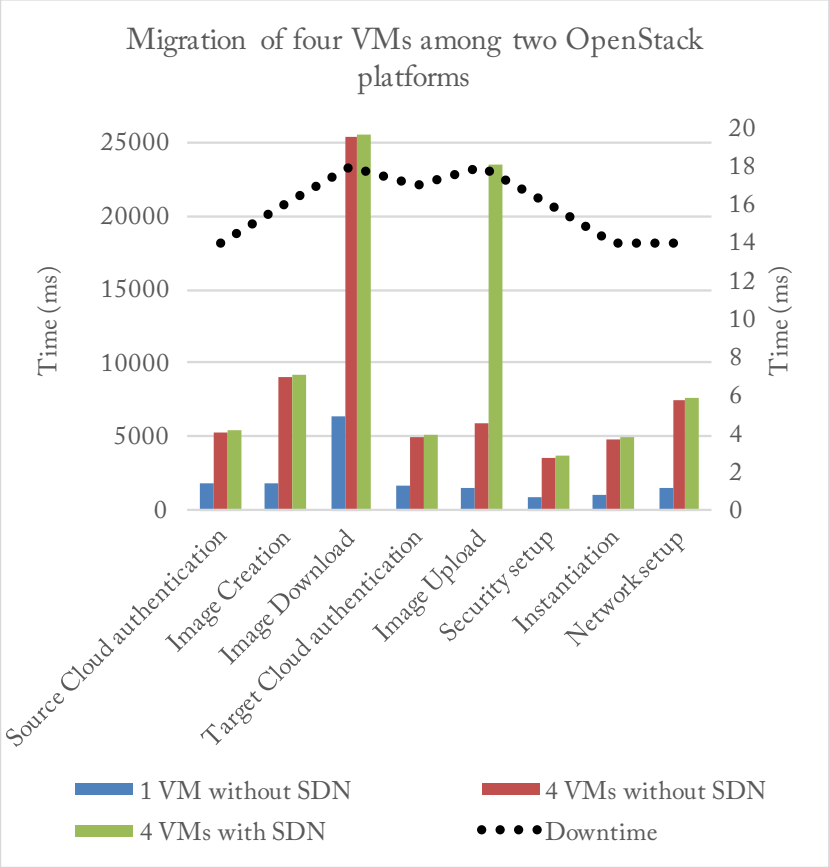

Figure 7: VM Migration times

(i) One VM without the SDN configuration: In this case we migrate one VM without SDN to serve as a fundamental benchmark.

(ii) Four VMs without the SDN configuration: In this case we migrate four VMs between two OpenStack systems and we measure the average time of all VMs as a whole.

(iii) Four VMs with the SDN configuration: In this case we migrate four VMs and we perform the SDN configuration.

(iv) Downtime: We measure the downtime during SDN setup (demonstrated by the right y axis).

Based on this observation we conclude to the following results. The downtime is considered significant low compared to the times spent from OpenStack for configuration of the various steps. The difference between four VMs with and without SDN is almost similar, thus SDN does not affect the performance of the overall inter-cloud mediation service.

7. Conclusions

This work focused on designing cloud applications using an SDN architecture to eliminate static endpoints and to allow dynamic connectivity among VMs. A cloud application can communicate with third party services in order to integrate its whole functionality. Thus, in case of moving the application to a different platform (e.g. due to new requirements or better SLAs) the migration process becomes a hurdle. Another similar case is the migration of multi cluster deployment of a system (e.g. distributed databases) in a cloud provider. The migration of such systems requires re-configuration of the network settings (e.g. interfaces, MAC addresses etc.) of the VMs, an action that decomposes the VM cluster. We suggested that by braking down the network configuration into tractable components we can setup network step by step. The proposed architecture is flexible and dynamic and allows easily reconfiguration of VMs without loosing their internal connections. The experimental analysis demonstrates a series of different cases so to extract benchmark features for future studies.

This includes inter-cloud mobility times for VM migration among OpenStack platforms to demonstrate the various steps of migration and the total times required to create a new running instance (that will be a clone of the original VM). We also migrate different sizes of VM images in order to explore how VM size affects migration times. This includes the SDN controller

configuration times with regards to the time to create the SDN bond among VMs, the times to update the SDN controller (when a migration has occurred) and the downtime during the update process. We also will focus on the live portability use case where we will utilize a load balancing solution to migrate a cluster of VMs among clouds. Also, we aim to explore different experimental configurations and demonstrate heterogeneous VM migrations.

8. Acknowledgment

This work was supported *DataWay - Real-time Data Processing Platform for Smart Cities: Making sense of Big Data* grant of the Romanian National Authority for Scientific Research and Innovation, CNCS - UEFISCDI, project number PN-II-RU-TE-2014-4-2731.

9. References

[1] C. S. A. Mladenow, N. Kryvinska. Towards cloud-centric service environments. *Springer, The Society of Service Science, Journal of Service Science Research*, 4(2):213–234, 2012.

[2] S. Al-Kiswany, D. Subhraveti, P. Sarkar, and M. Ripeanu. Vmflock: Virtual machine co-migration for the cloud. In *Proceedings of the 20th International Symposium on High Performance Distributed Computing*, HPDC '11, pages 159–170, New York, NY, USA, 2011. ACM.

[3] R. Bradford, E. Kotsovinos, A. Feldmann, and H. Schiöberg. Live wide-area migration of virtual machines including local persistent state. In *Proceedings of the 3rd International Conference on Virtual Execution Environments*, VEE '07, pages 169–179, New York, NY, USA, 2007. ACM.

[4] A. Castiglione, F. Palmieri, U. Fiore, A. Castiglione, and A. De Santis. Modeling energy-efficient secure communications in multi-mode wireless mobile devices. *J. Comput. Syst. Sci.*, 81(8):1464–1478, Dec. 2015.

[5] B. Cully, G. Lefebvre, D. Meyer, M. Feeley, N. Hutchinson, and A. Warfield. Remus: High availability via asynchronous virtual machine replication. In *Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation*, NSDI'08, pages 161–174, Berkeley, CA, USA, 2008. USENIX Association.

[6] U. Deshpande, X. Wang, and K. Gopalan. Live gang migration of virtual machines. In *Proceedings of the 20th International Symposium on High Performance Distributed Computing*, HPDC '11, pages 135–146, New York, NY, USA, 2011. ACM.

[7] C. Esposito, M. Ficco, F. Palmieri, and A. Castiglione. Interconnecting federated clouds by using publish-subscribe service. *Cluster Computing*, 16(4):887–903, 2013.

[8] D. Ferrazani Mattos and O. Muniz Bandeira Duarte. Xenflow: Seamless migration primitive and quality of service for virtual networks. In *Global Communications Conference (GLOBECOM), 2014 IEEE*, pages 2326–2331, Dec 2014.

[9] S. Ghorbani and M. Caesar. Walk the line: Consistent network updates with bandwidth guarantees. In *Proceedings of the First Workshop on Hot Topics in Software Defined Networks*, HotSDN '12, pages 67–72, New York, NY, USA, 2012. ACM.

[10] S. Ghorbani and B. Godfrey. Towards correct network virtualization. *SIGCOMM Comput. Commun. Rev.*, 44(4):–, Aug. 2014.

[11] S. Ghorbani, C. Schlesinger, M. Monaco, E. Keller, M. Caesar, J. Rexford, and D. Walker. Transparent, live migration of a software-defined network. In *Proceedings of the ACM Symposium on Cloud Computing*, SOCC '14, pages 3:1–3:14, New York, NY, USA, 2014. ACM.

[12] C.-Y. Hong, S. Kandula, R. Mahajan, M. Zhang, V. Gill, M. Nanduri, and R. Wattenhofer. Achieving high utilization with software-driven wan. In *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, SIGCOMM '13, pages 15–26, New York, NY, USA, 2013. ACM.

[13] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu, J. Zolla, U. Hölzle, S. Stuart, and A. Vahdat. B4: Experience with a globally-deployed software defined wan. *SIGCOMM Comput. Commun. Rev.*, 43(4):3–14, Aug. 2013.

[14] M. Jammal, T. Singh, A. Shami, R. Asal, and Y. Li. Software-defined networking: State of the art and research challenges. *CoRR*, abs/1406.0124, 2014.

[15] E. Keller, S. Ghorbani, M. Caesar, and J. Rexford. Live migration of an entire network (and its hosts). In *Proceedings of the 11th ACM Workshop on Hot*

Topics in Networks, HotNets-XI, pages 109–114, New York, NY, USA, 2012. ACM.

[16] E. Keller, J. Rexford, and J. Van Der Merwe. Seamless bgp migration with router grafting. In *Proceedings of the 7th USENIX Conference on Networked Systems Design and Implementation*, NSDI'10, pages 16–16, Berkeley, CA, USA, 2010. USENIX Association.

[17] T. Koponen, K. Amidon, P. Balland, M. Casado, A. Chanda, B. Fulton, I. Ganichev, J. Gross, N. Gude, P. Ingram, E. Jackson, A. Lambeth, R. Lenglet, S.-H. Li, A. Padmanabhan, J. Pettit, B. Pfaff, R. Ramanathan, S. Shenker, A. Shieh, J. Stribling, P. Thakkar, D. Wendlandt, A. Yip, and R. Zhang. Network virtualization in multi-tenant datacenters. In *Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation*, NSDI'14, pages 203–216, Berkeley, CA, USA, 2014. USENIX Association.

[18] H. A. Lagar-Cavilla, J. A. Whitney, A. M. Scannell, P. Patchin, S. M. Rumble, E. de Lara, M. Brudno, and M. Satyanarayanan. Snowflock: Rapid virtual machine cloning for cloud computing. In *Proceedings of the 4th ACM European Conference on Computer Systems*, EuroSys '09, pages 1–12, New York, NY, USA, 2009. ACM.

[19] H. H. Liu, X. Wu, M. Zhang, L. Yuan, R. Wattenhofer, and D. Maltz. zupdate: Updating data center networks with zero loss. *SIGCOMM Comput. Commun. Rev.*, 43(4):411–422, Aug. 2013.

[20] S. Lo, m. Ammar, and E. Zegura. Design and analysis of schedules for virtual network migration. In *IFIP Networking Conference, 2013*, pages 1–9, May 2013.

[21] V. Medina and J. M. García. A survey of migration mechanisms of virtual machines. *ACM Comput. Surv.*, 46(3):30:1–30:33, Jan. 2014.

[22] D. Penzel, N. Kryvinska, C. Strauss, and M. Gregu. The future of cloud computing: A swot analysis and predictions of development. In *Future Internet of Things and Cloud (FiCloud), 2015 3rd International Conference on*, pages 391–397, Aug 2015.

[23] S. Rajagopalan, D. Williams, H. Jamjoom, and A. Warfield. Split/merge: System support for elastic execution in virtual middleboxes. In *Proceedings of the 10th USENIX Conference on Networked Systems Design and Implementation*, nsdi'13, pages 227–240, Berkeley, CA, USA, 2013. USENIX Association.

[24] C. P. Sapuntzakis, R. Chandra, B. Pfaff, J. Chow, M. S. Lam, and M. Rosenblum. Optimizing the migration of virtual computers. *SIGOPS Oper. Syst. Rev.*, 36(SI):377–390, Dec. 2002.

[25] S. Sotiriadis and N. Bessis. An inter-cloud bridge system for heterogeneous cloud platforms. *Future Generation Computer Systems*, (0):–, 2015.

[26] S. Sotiriadis, N. Bessis, A. Anjum, and R. Buyya. An inter-cloud meta-scheduling (icms) simulation framework: Architecture and evaluation. *Services Computing, IEEE Transactions on*, DOI: 10.1109/TSC.2015.2399312(99):1–1, 2015.

[27] S. Sotiriadis, N. Bessis, and N. Antonopoulos. Towards inter-cloud schedulers: A survey of meta-scheduling approaches. In *Proceedings of the 2011 International Conference on P2P, Parallel, Grid, Cloud and Internet Computing*, 3PGCIC '11, pages 59–66, Washington, DC, USA, 2011. IEEE Computer Society.

[28] S. Sotiriadis, N. Bessis, P. Kuonen, and N. Antonopoulos. The inter-cloud meta-scheduling (icms) framework. In *Proceedings of the 2013 IEEE 27th International Conference on Advanced Information Networking and Applications*, AINA '13, pages 64–73, Washington, DC, USA, 2013. IEEE Computer Society.

[29] L. Vacanas, S. Sotiriadis, and E. Petrakis. Implementing the cloud software to data approach for openstack environments. *Adaptive Resource Management and Scheduling for Cloud Computing, Held in conjunction with PODC-2015*, 2015.

[30] K. Ye, X. Jiang, R. Ma, and F. Yan. Vc-migration: Live migration of virtual clusters in the cloud. In *Proceedings of the 2012 ACM/IEEE 13th International Conference on Grid Computing*, GRID '12, pages 209–218, Washington, DC, USA, 2012. IEEE Computer Society.