Horizon Project

ANR call for proposals number ANR-08-VERS-010
FINEP settlement number 1655/08

Horizon - A New Horizon for Internet

WP4 - TASK 4.3: Implementation of a Demonstrator
(Annex L)

Institutions

Brazil	**France**
GTA-COPPE/UFRJ	LIP6 Université Pierre et Marie Curie
PUC-Rio	Telecom SudParis
UNICAMP	Devoteam
Netcenter Informática LTDA.	Ginkgo Networks
	VirtuOR

Contents

List of Figures

Chapter 1

Introduction

The Internet has evolved and deeply changed in the last 40 years. Currently, the network presents many vulnerabilities and challenges for supporting innovation in the network core. Besides, the network management by human administrators became costly and prone to failure, and the simple automation of management through software components may worsen the problem due to the wide variety of systems and unexpected behaviors. These are the main reasons for condemning the Internet current architecture and developing a new one. The goal of Horizon Project [1] is to conceive and test a new architecture for a post-IP environment. The key features to design and build this new architecture are the network virtualization and the piloting system, which cope with the network constraints. This architecture is intelligence-oriented and based on multi-agent systems in order to guarantee an efficient and scalable network management. Besides, it should be easily implemented, scalable, secure, robust, and should also provide quality of service on demand. Choosing the virtual network to be used depends on the context and on the service to be offered.

To reach these goals, we developed an autonomic-oriented architecture to support self-organized, self-control, and self-secure management. This architecture is based on the piloting plane, which automatically chooses the best parameters to optimize the behavior of the network. Indeed, the autonomic-oriented architecture associates to each network device a situated view used to determine the context and to choose and optimize control algorithms and parameters.

Another important concept for post-IP networking used in this project is the network virtualization, which allows the abstraction of the network as virtual substrate. A virtual substrate represents a coherent functional group of instances of virtual routers rather than physical routers. The architecture proposed in Horizon supports different kinds of virtual networks running over the same physical substrate. The developed control and management system guarantees the efficient use of physical resources, as well as an adequate mapping of the virtual resources over the physical resources. Hence, the proposed architecture provides a background for new environments concerning wireless networks, sensor networks, delay tolerant networks, or RFID networks. Moreover, the proposed architecture also provides the basis for supporting new proposals for mobility [2], security [3, 4], and content distribution [5, 6], also in the context of Future Internet.

In this dynamic multi-stack network, the piloting system distributes physical resources among virtual networks. This allows a service provider to simultaneously run multiple end-to-end services with different qualities of service and different securities. Virtual networks are created and destroyed when necessary. The piloting system associated to network virtualization allows a better use of physical resources, adapting virtual networks to the customers.

The teams on Horizon Project developed the basis for a complete piloting plane to control virtualized environment. Self-piloting is used to facilitate continuous tuning of the virtual networks, adaptation to unpredictable conditions, and prevention and recovery from failures. The self-piloting scheme feeds control algorithms with specific information collected with the developed monitoring modules. The goal of the developed piloting plane is to obtain the knowledge required to optimize the creation and the destruction of virtual networks and the distribution of the physical resources. The developed solution considers aspects such as security, QoS, reliability, robustness, context, access, service support, self-management, and self control of the communication resources and services. Therefore, the project Horizon provided ways for transitioning from a static mono-stack Internet to a dynamic multi-stack network.

As a summary, this project designed and developed a post-IP architecture using a piloting plane and virtual networking. To tackle this challenge, multi-agent systems were used as a modeling foundation. Hence, the proposed architecture supports human management as well as artificial agents that work as assistants, problem solvers, planners, etc., using various types of interaction and coordination. The multi-agent paradigm was selected as the basis for our solution due to some intrinsic properties of agents, such as autonomy, proactivity, adaptability, cooperating, and mobility. Moreover, the notions of agents and organizations and their decentralized and proactive nature match well the requirements of large-scale pervasive computing environments.

1.1 Report outline

This deliverable presents the results of the Task 4.3 and the final conclusions of the project. This task, based on the requirement analysis produced by Task 4.1 and on the integration models described in Task 4.2, has the objective of designing the overall virtualized and autonomous infrastructure with the piloting system. Task 4.3 presents the final integration results, as well as the demonstration of some of the main modules developed in Horizon. Hence, this report presents the ultimate Horizon architecture.

This report addresses the following themes. In Chapter 2, we describe the Horizon general proposal structure, as well as the developed solutions for each item. After, in Chapters 3, 4, and 5, we describe some of the last developed modules, which provide a full integration of the proposed architecture.

Chapter 3 describes a common interface for managing virtual networks built over different virtualization platforms. Network virtualization is a long-term challenge due to the inherent difficulty to virtualized I/O devices. There are different proposals that virtualized networks at different levels such as Xen, which virtualizes computers, and OpenFlow, which virtualizes network flows. As networks evolve, new scenarios with hybrid virtualization approaches may appear, introducing a new challenge, which is to allow an efficient management interface of virtualized networks despite of its individual components and heterogeneous underlying technologies. In this chapter we describe the Substrate Abstraction for Virtualized Environments (SAVE), which is a unified approach to manage virtual networks independently of its underlying substrate and fills a gap in the management of virtual networks. We show that a standardized interface allows the proper management of the virtualized networks, providing a complete interface for network management for the piloting plane. We perform some experiments to show that the parameters of different virtualization platforms can be used to generate generic virtual network control and management parameters.

We also extended the agent modules described in the previous reports in Chapter 4. Autonomic networks were proposed to deal with management problems by enabling systems to self-manage. But, in order to perform a self-management in an optimal, robust and secure way, it is necessary to have a piloting system. The main goal of a piloting system is to regulate and adapt the virtual network in response to changing context in accordance with applicable high-level goals and policies. In this context, this report extends the self-organizing and normative piloting system to govern the entities of the network in a decentralized way. Moreover, we provide a simulation environment that enable users to experiment and observe the network behavior in face of the application of different normative and organizational configurations of the piloting plane.

Next, Chapter 5 presents an architecture for the deployment of clouds over virtualized networks. This architecture, conceived within the scope of the Horizon architecture, was used to build a software infrastructure. This infrastructure is composed of a network substrate, a set of software tools for creating virtual networks on demand (autonomic self-management system), a computational grid, and a workflow management system.

To provide a better comprehension of the final Horizon architecture, in Chapter 6 we describe some demonstrations that can be performed using the developed modules. For instance, we provide a prototype to experiment the use of virtual routers and the Ginkgo platform, when using image processing workflows, utilized in e-Science applications. We also describe demonstrations using the virtualization tools and the piloting plane simulator. Finally, Chapter 7 presents the main conclusions of this work and future directions.

Chapter 2

Horizon general view

The objective of Horizon project is to define an architecture from the data plane to the knowledge plane via the virtualization, the management, the control, and the piloting planes. Most of the efforts applied in this project were focused on the virtualization and the piloting planes. The virtualization will provide several forwarding planes where different flows will be allocated depending on control algorithms fed by the piloting plane. The piloting plane introduces a smart process for optimizing partitioning in the physical resources and feeding the control algorithms. This architecture also concerns about fulfilling the user requirements. Therefore, the piloting plane is defined with strong intelligence schemes and is able to react in real time.

The project was divided into four work packages:

1. **The context-aware post-IP architecture (Deliverables D1.1, D1.2, and D1.3 [7])** - In the first steps of the project, we determined the main features to develop an autonomic post-IP environment. Post-IP networks present specific challenges, because, up to now, we have no definition about what should be the new protocol stack for the network, or even how the IP could be modified to provide new functionalities without impacting on the network performance. Hence, the architecture developed by Horizon should provide support to any kind of protocol stack that could be developed as an option to IP. One important characteristic that were observed was that networks must be aware of their context in order to provide an efficient control and management. Indeed, the context provides to each node a 'situated cognition' about the network state. This way, the node is able to collect and store only the subset of measures that concerns to its duties when managing the network. Hence, in this work package, we observed the main requirements for building a context-aware architecture for a

new Internet.

2. **The virtual network environment (Deliverables D2.1, D2.2, and D2.3 [7])** - A second working area within Horizon was the development of virtualized environments that presented high packet forwarding performance and high isolation among virtual slices. Virtual networking enables the sharing of the network equipments between several entities in a way that the decisions and the network usage of an entity do not interfere with the decisions and usage of the other entities sharing the same substrate. Virtual networking allows that the physical resources are better used, since logical network and physical equipments are decoupled by the virtualization layer. Hence, it is possible to change a logical router between different physical routers to make a preventive maintenance or to reduce power consumption. Although this is an important feature, it is not the main outstanding characteristic provided by network virtualization. Indeed, network virtualization allows that different proposals for networks co-exist over the same physical substrate. Hence: (i) using network virtualization, there is no need to find a single solution to all Internet problems; (ii) virtualization is the path for transitioning the current Internet to a new architecture, because both environments could exist until all devices were updated to the new version of the protocol stack; (iii) network virtualization is also the natural path for testing new proposals. Using this technique, it is possible to verify in large scale if a new proposal is scalable and correctly implemented. This leads us to another characteristic of virtual environments, which is the elastic resource allocation. Using virtualization, it is possible to build a new solution in a small environment and only extend this environment if the new solution demands more resources. This avoids high initial costs with infrastructure and also losses due to an over or underestimation in the equipment requirements. Hence, network virtualization is a key feature for supporting innovation in the network core.

3. **The piloting plane (Deliverables D3.1, D3.2, and D3.3 [7])** - The developed piloting system, used to control and manage the virtual networks, can be seen as an aggregation of two specific planes: a knowledge plane and an orchestration plane. The knowledge plane is in charge of recovering the knowledge useful for feeding the control and management algorithms. The orchestration plane, on the other hand, is in charge of indicating the course of the virtual network. The advantage of the piloting system is the possibility to adapt in real time

through the management and orchestration plane. The piloting process aims to adapt the virtual network to new conditions and to take advantage of the intelligent decisions to alleviate the global network. Therefore, the role of the piloting system is to govern and adapt the virtual network in response to changing context in accordance with applicable high-level goals and policies. It supervises and integrates all other planes' behavior, ensuring integrity of management and control operations. In this context, the use of a multi-agent system permits the achievement of a more attractive orchestration process due to the following points: (1) each agent holds different processes (dynamic planners, low coupling); (2) the agents are cooperative and reactive, in the sense that they are able to use a privileged view of their neighbors and individual knowledge together. As mentioned, the purpose of the piloting system is to regulate and integrate the behaviors of the network in response to changing context and in accordance with applicable high-level norms. Norm is a regulation mechanism that defines a set of rules to the system agents in order to ensure a social order that enables the achievement of the global goal of the organization. Our piloting system can be seeing as a self-organizing control framework into which any number of network devices can be plugged into or out of in order to achieve the required service level agreement. Therefore it hosts several self-organizing piloting systems each one managed by a piloting agent. Each agent maintains its own knowledge base consisting of a set of data models about the physical and virtual devices. In this way, agents manage virtual devices by following a set of norms and using a set of knowledge. Moreover, agents can communicate and cooperate with each other by using behaviors.

4. **Piloting the virtual networking environment (Deliverables D4.1, D4.2, and D4.3 [7])** - This final work package presents as main functionality the integration of all the proposed solutions developed in the other work packages. The objective of the Work Package 4 is to create the environment of the piloting system and to supervise the work to be carried out in WP2 (Virtualization) and WP3 (The Piloting plane). Besides, the WP4 specifies the service control requirements establishing the basis for the WP activities in a form of policy based architectures or prototypical applications. Hence, in this work package we specified some of the main requirements of the current Internet and we proposed some case studies to evaluate the proposed solutions. We also designed the overall architecture of Horizon, which considered a local and a global view. In the local view, we developed and integrated

local control and management functions. Hence, we developed mechanisms that run inside each node in the network. In the global view, we developed and integrated algorithms to control the network as whole, considering the observations made by all nodes. In this sense, we developed both centralized and decentralized solutions, considering the use of agents to monitor network nodes and to exchange relevant information, considering the situated view of each network element. In this task, we also prepared demonstrators, to simplify the comprehension of the developed work. Some of these demos are also available through videos, available in the project homepage [1]. They can be reproduced by downloading the developed software and installing then according the corresponding manuals in an easy way.

Therefore, in this project we used a methodology that has focused in two main features: the development of an agent-based autonomic piloting system and of a framework for using virtual networks. The last work package objective, described in this report, is to fuse both areas in a consistent prototype that proves the main paradigms proposed in this project. Hence, we developed a common interface to provide network management and control for virtual networks using any virtualization platform (Chapter 3) and also extensions to our autonomic and agent-based environment (Chapters 4 and 5). Finally we developed testbeds and demonstrators to provide an easy experimentation of the integration of the project results.

Chapter 3

Common interface for control and management of virtual networks

Network virtualization is an approach proposed to increase the network core support for innovation, which allows that multiple networks coexist over the same physical substrate [8]. Hence, virtual networks share computational resources and execute different protocol stacks without interfering with each other. In this scenario, the ISP role is decoupled into different levels of service provision [9]. There are infrastructure providers, which manage physical networks, the virtual network providers, which allocate slices for virtual networks over different physical infrastructure domains, and the network operators, which contract virtual network providers to deploy their services.

The virtualization technique allows the abstraction of computational resources, so multiple logical environments can execute and share the same physical infrastructure. At the same time that network virtualization enables innovation, it introduces new problems, mainly related to the operation of virtual networks and how to share resources and keep isolation [10]. Indeed, the management of virtual networks faces challenges, because virtualization technologies available today differ deeply in their architectures [11]: some are based on the notion of complete virtual machines, such as Xen [12], others in the concept of containers, such as OpenVZ [13], while others are based in a completely different approach, such as OpenFlow, in which the network control is centralized and the data plane is shared by all virtual networks [14]. Another aspect of the network virtualization is the emerging cloud computing scenario [15]. In cloud computing, data centers use virtualization to slice its resources among their customers, each with different requirements. Different clouds can be interconnected and each cloud vendor can implement its own virtualization solution. For instance, a virtual network operator can contract virtual network elements belonging to different cloud providers and use service providers to interconnect them.

Virtual network operators should not deal with the various subtleties of the different network virtualization techniques. Indeed, the virtual network operators should specify virtual networks only in terms of virtual topology and node capacity. The management of the different virtual network platforms and the mapping of the abstract management primitives into the real primitives, which are platform specific, should be done by underlying tools of the virtual network provider.

We propose a common interface for managing virtual networks deployed over different virtualization platforms called the Substrate Abstraction for Virtualized Environments (SAVE). Since it is hard to provide a single interface to manage all specific parameters of all virtualization technologies, we define an abstraction level that specifies the main management primitives and virtual network dimensions. SAVE takes into consideration all the different underlying virtualization substrates and allows network operators to simply operate networks despite of their particular characteristics.

The main contributions of SAVE are summarized as follows:

- SAVE introduces an abstraction layer that allows the management of virtual networks composed of virtual routers and virtual servers created with different platforms;

- SAVE defines a set of dimensions which describe any virtual network element as well as high-level primitives to manage any virtual network; and

- the architecture of SAVE is designed to be extensible to new virtualization platforms and to allow the support of multiple virtualization solutions at the same time.

Our proposal was implemented and tested, providing compatibility to Xen and OpenFlow platforms. We developed the modules that model the management primitives of these two platforms into the defined high level abstraction layer. Experiments show that the proposed mapping scheme guarantees a high conformance between the values contracted by the virtual network operator and the resources provided by the infrastructure provider.

3.1 State of art

The development of an interface for configuring virtual machines created under different technologies has been a growing concern in the last years. Virtual machines, besides being used in data centers, are also the substrate for creating virtual networks. The purpose of a virtual network is to provide a "soft networking environment", which means flexible resource management, configurable topology, and programmable network architecture [16]. Managing all this parameters simultaneously in different virtualization platforms, however, is a huge challenge.

Libvirt is a virtual machine management library that provides an interface to different hypervisors [17]. This library provides primitives such as turning a node on and off, migration, and monitoring. Another similar approach was proposed by Peng et al., which provides virtual machine management functions based on an agent platform [18]. Although these approaches provide primitives for different hypervisors, they do not deal with virtualized network functions, such as setting virtual network topologies. Moreover, these approaches do not intend to create a common interface for any virtualization platform, but a set of important primitives of each platform.

Leon-Garcia et al. developed a system based on the MIBlet concept for effectively designing and managing virtual networks [16]. The key idea is to logically split the Management Information Base (MIB) of a network element into multiple MIBs, called MIBlets. The resource agent running in each network node then provides a selective view of the element to each customer network resource management system, which is responsible for managing the virtual network. Then, the management system controls the access of the customers to the physical nodes, while also offers abstract and selective views of the physical network resources allocated to the virtual network. This system creates partitions in the physical shared resources, which are the ports, the address space, and the bandwidth, by using the MIBs as an open interface for managing network elements. This proposal was developed to work in ATM networks using commercial network equipment, but it does not provide an interface to deal with virtual networks created with Xen, OpenFlow, and other network virtualization platforms.

Building federated systems is another way to integrate different network virtualization platforms. Park et al. present a framework to federate and operate virtualized networks [19]. The authors propose a common federation interface that provides three key functionalities, which are the representation of the network core schema, operational data exchanges, and federated network operations. The authors do not explain, however, how to map this common interface into different administrative domains and virtualization platforms.

Virtual network environments can be created in a single physical node for testing new mechanisms or new network configurations. Virtualization tools such as NetKit [20] and Manage Large Networks (MLN) [21] model virtual networks as a whole and provide mechanisms for creating and deploying the virtual networks. These proposals, however, are coupled to specific technologies. Galan et al. propose a generic management model for virtual network environments created over a single physical machine that works over any virtualization platform [22]. The authors propose a modeling technique for virtualized infrastructures and provide a management interface for virtual network environments, based on the Common Information Model (CIM) Schema [23]. The CIM Schema presents extensions that models virtualization platforms such as Xen and VMware. The proposal of Galan et al. develops an extension called Virtual Network Environment CIM (VNE-CIM) that models parameters such as the life cycle and the resource allocation in virtual networks built in a single physical node. Although the management of this kind of virtual network is simpler than the management of a virtual network distributed over a set of physical nodes, these approaches introduce important concepts for defining virtual networks.

SAVE differs from all these approaches, because it provides a high level common interface which deals with the primitives for managing virtual networks. Different from approaches such as Libvirt, which are specific to virtual machine management, SAVE defines primitives for virtual networking. SAVE also proposes a common interface just as in the proposals of Leon-Garcia et al. and Park et al., but our approach supports different virtualization platforms and explains how to map the high level primitives into the different real primitives.

3.2 Network Virtualization

A virtual network is defined as a set of virtual nodes connected through virtual links, which are instantiated over a physical infrastructure as seen in Figure. 3.1. There, two virtual networks belonging to different owners are instantiated over a physical infrastructure managed with SAVE. These virtual network elements share physical resources with virtual elements from other networks. Nevertheless, virtual networks should remain isolated from each other and must respect service level agreements and fulfill the requirements of clients.

A problem that arises with network virtualization is how to define a subset of management operations that can be applied in generic virtualized archi-

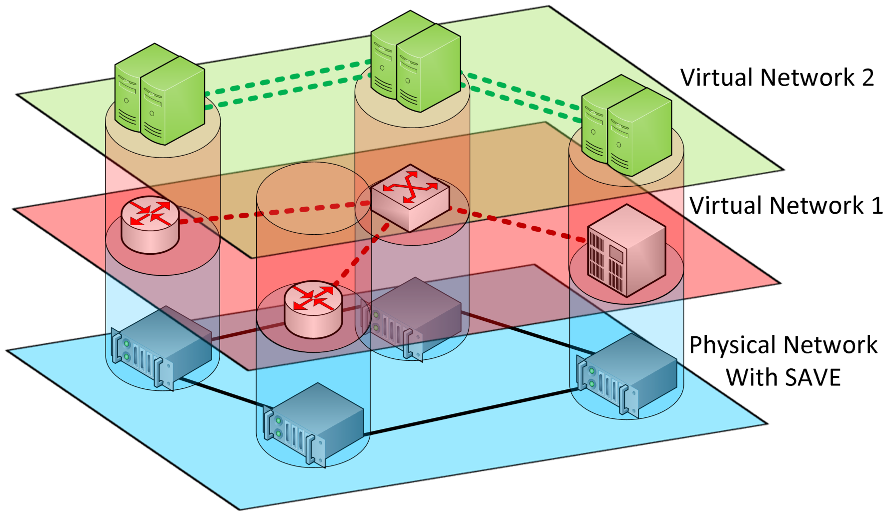


Figure 3.1: Example of two virtual networks instantiated over a physical infrastructure managed with SAVE.

tectures and also, which virtual dimensions can be controlled. Tools such as FlowVisor [14], VNEXT [24], OMNI [25], and others provide virtual network management primitives that are platform specific. Hence, instantiating nodes, creating virtual links, and even monitoring network status vary from platform to platform, because the way to define, access, and manage nodes is different.

3.3 SAVE Architecture

SAVE defines a set of primitives to easily manage and define a generic virtual network. We propose an architecture that gives the basis for implementing and expanding these primitives to any virtualization platform. Our architecture also provides a secure access interface for the virtual network operators.

The proposed architecture is composed of five modules as seen in Figure 3.2. The `network operator` represents any external entity that interacts with SAVE, such as a network manager or an artificial intelligence driven manager. The network operator accesses the `HTTP server`, which is the interface between operators and SAVE, to manage the virtual network. The `HTTP Server` receives the network operator's request and forwards them to the `controller`, which consults the `access control list` to verify the operator permissions to manage a given network. After that, the `controller` accesses

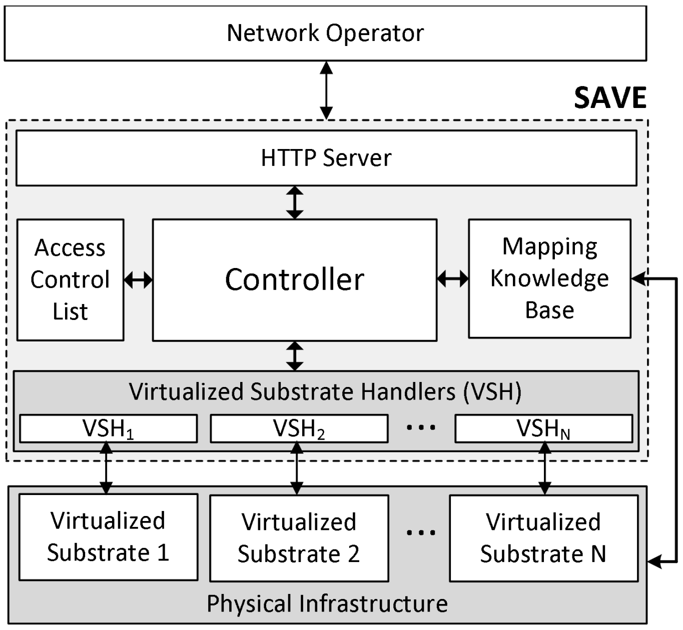


Figure 3.2: The SAVE architecture and its main modules: HTTP Server, Access Control List, Controller, Handlers and Mapping Knowledge Base.

the `mapping knowledge base`, which stores the mapping procedures for each virtualization technology. Hence, this module provides the knowledge of how to convert high level primitives into real management primitives of each virtualization platform. This module also stores the mapping of a given virtual topology in the real infrastructure. Hence, the `mapping knowledge base` informs the controller which network virtualization platforms are being used by the network operator and how to convert the high-level requests into real requests in each platform. Given that, the `controller` maps the parameters and send the requests to the adequate `virtualized substrate handlers`.

3.3.1 HTTP Server

The `HTTP Server` provides services to the network operators to retrieve network status and configure networks parameters by using the high-level control primitives. Hence, this module works as common interface that creates the abstract level perceived by the network operators. Indeed, the network operator has no knowledge about the infrastructure providers and their virtualization technologies.

The `HTTP Server` is based on the web service approach and delivers security and access control. Connections among network operators and the server are encrypted and ensure proper end-to-end data-exchange confidentiality. The network operators can belong to different access lists, which defines which information can be seen by each operator and which management operations can be performed. Control and monitoring requests and answers are exchanged as XML requests between the server and the network operators.

3.3.2 Controller

The `controller` is the central module of SAVE. It receives network operator requests, verifies operator credentials, translate the request in substrate specific operations and dispatch the requests to the `handlers`. For example, suppose that a network operator wants to instantiate a new node in a given virtual substrate. The `controller` verifies with the `mapping knowledge base` the current virtualization technologies being used in the virtual network and how the node instantiation parameters are mapped in those virtualization technologies. After that, the `controller` generates a request containing the identifier of the virtual network and the substrate specific parameters to fulfill the demands of the operator. The request is forwarded to the proper `virtualized substrate handler` and finally the virtual node is instantiated.

3.3.3 Access Control List

The `access control list` contains associations among network operators and virtual networks with the corresponding permissions. The module enhances management security and ensures that only operators with correct credentials can perform actions in the virtual networks. For example, a virtual network owner can give monitoring permissions to all of its operators, and allows only the network manager to perform virtual network topology changes.

3.3.4 Mapping Knowledge Base

The `mapping knowledge base` is divided in sub modules. The `substrate base` sub module continuously monitors the virtualized substrates to detect the current state of virtual networks. The information obtained in the module is stored in the `mapping database` sub module. The `virtual substrate mapping` modules are specific for each virtualized substrate technology, containing intelligence about primitive mapping on each different substrate. If a

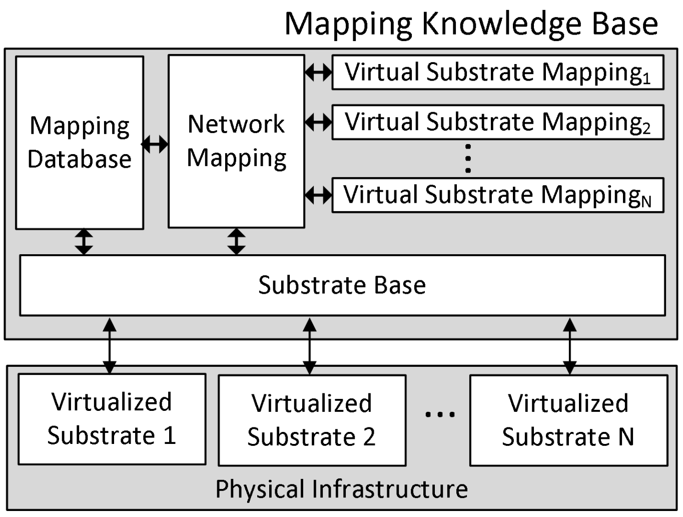


Figure 3.3: Mapping Knowledge Base. Stores virtual network information and mapping procedures.

new substrate technology is created, we can add support to it by just adding a new mapping sub module specific to the new technology. Finally, the `network mapping` sub module exchanges information with the `substrate base`, the `mapping database` and the `virtual substrate mapping` sub modules to translate network operator requests into specific parameter mappings for each substrate. This information is further received by the `controller` module.

3.3.5 Virtualized Substrate Handler (VSH)

`Handlers` allow SAVE to communicate with virtualized substrates. They develop the role of a translator, receiving management operations and virtual network parameters from the `controller` and issuing instructions to the corresponding virtualization substrate. Hence, the controller maps high level primitives into the primitives of each substrate and the VSHs map these primitives into small ones provided by other tools. For instance, if a virtual network operator changes its topology by adding a new node and the corresponding virtual links, the controller will receive, process, and forward this request to the corresponding VSH. The VSH will then translate this request into small platform specific ones provided by tools such as Libvirt or VNEXT, which are create a node, configure the node parameters, change allocation parameters, and create virtual links.

3.4 SAVE Network Dimensions

SAVE considers a set of network characteristics, or dimensions, which are enough to define a virtual network. Other works propose different sets of characteristics, but these sets are not always platform independent [14, 23]. The idea of SAVE is to define a set of dimensions which are platform independent and that describe all the most important network characteristics used in network control and management.

The proposed dimensions are divided into two main subsets, which are the interconnection dimensions and element dimensions. The interconnection dimensions describe the characteristics that define the virtual network. The element dimensions allow the definition of specific characteristics for each network element.

3.4.1 Interconnection Dimensions

The interconnection dimensions define characteristics that are common to any node in the virtual network and that specify the network as a whole. Hence, these dimensions comprise the network topology, which depends on the element dimensions, and the traffic definition.

3.4.1.1 Topology

The virtual network topology defines how virtual nodes and virtual links are connected. The definition of both the node and the links is specified in the element dimensions. Hence, topology interprets different nodes as undistinguished elements with different identifiers. The same applies for the links.

The main virtual network operator actions correlated to the topology dimension are to create, delete, modify, monitor, and view the topology. The network view and monitoring allows the virtual network control to react to events such as link failures and loops. The other actions allow the operator to design and maintain the virtual network.

It is important to notice that the operator specifies characteristics to the nodes and the links, but the operator has no control over the map of virtual elements into physical elements, except for the physical location of the network inputs and outputs. Hence, one hop link between two virtual nodes in a virtual topology may be mapped into multiple physical hops, as long as the requirements of virtual node and links are maintained. Hence, the virtual network monitoring is harder to be implemented than the physical one, because a group of measurements in different nodes and/or links must be aggregated to provide a view of a single measurement to the virtual network operator.

3.4.1.2 Traffic Definition

Many virtual networks share the same physical substrate. Hence, the infrastructure providers must receive a description of the traffic that belongs to each virtual network in order to provide correct packet forwarding through the right virtual elements. Nevertheless, each infrastructure provider works with a different virtualization platform, each with different definitions of the traffic. For instance, OpenFlow networks define virtual network traffic according to all header fields of the TCP/IP model. Xen, OpenVZ, and other similar platforms use the IP-based packet forwarding primitives, which are based only on the destination IP address. Moreover, a virtual network operator may demand a protocol stack completely uncorrelated with TCP/IP. Thus, the role of SAVE is to give support to the virtual network provider to translate the traffic definition of the virtual network operator into traffic definitions of the infrastructure provider.

Since virtual networks can use any kind of protocol stack, it is not possible to specify the traffic of each virtual network based on the TCP/IP headers. Hence, SAVE uses two aspects for defining the traffic of each virtual network. The first aspect is the definition of the values of the bits of a packet. Each bit can be set as '0', '1', or 'x' (don't care). The 'x' bits work as a wildcard, which means that the bit can be '1' or '0'. The second aspect is the traffic sources. The virtual network operator must define which nodes in the virtual topology are traffic sources. This is important, because different virtual networks may have a correlated bit definition, but since they present different sources, their traffic can be differentiated. Using these two parameters, a large range of virtual networks can be defined.

Each infrastructure provider may classify the traffic in a different way. One may choose hardware-based packet classification, while other may choose to set a special header after classifying packets in the source nodes. These techniques, however, are not in the scope of SAVE.

3.4.2 Element Dimensions

The element dimensions define the specification of nodes and links of the virtual topology. Each virtual node in SAVE, besides having an identification, is defined based on the processing power, memory, and disc. A virtual link, which also has identification, is defined based on its bandwidth and on the end nodes connected by it.

3.4.2.1 Node Dimensions

A SAVE virtual node is defined by:

- **Processor -** a virtual processor is a slice of the physical processor. Network nodes use processor power to forward and process packets, besides updating network statistics. Without proper processor slicing, a given network can face packet drops and failures in the convergence of routing algorithms. Depending on the characteristics of the virtual node, the processing power has different means. When the virtual element simply forwards packets, the processor dimension means packet forwarding rate. When a virtual element is a middle box, it must execute specific processing as well, thus the processor dimension must involve cycles per second allocated to the virtual element. Thus, processor must be specified in terms of forwarding rate and cycles per second ;

- **Memory -** a virtual node uses memory for forwarding and processing packets. Also, the nodes use the memory for storing forwarding and filtering rules. Depending on the type of network element, the amount of allocated memory is different. Forwarding virtual elements must have enough memory to execute its operational system, its routing algorithms and to store routing tables. Virtual switches must store forwarding tables and forwarding rules. SAVE converts the memory allocation requests of the virtual network operators into the adequate amount of memory according to the kind of physical node used to create the virtual node ;

- **Disk -** a virtual node my require disk for storing monitored data and packet processing behaviors. Hence, SAVE allows the definition of an amount of non-volatile memory in each node.

3.4.2.2 Link Dimensions

A network element may specify bandwidth requirements through SLAs. Hence, each link connecting a node with other nodes must be described in terms of end-points and bandwidth. The bandwidth dimension is controlled in terms of bytes per second per virtual link. Based on this two data, SAVE is able to call a virtual topology mapping primitive, which will search for paths between the two virtual nodes that accomplish the bandwidth requirements set by the virtual network operator.

Table 3.1: Metrics of network elements

Element	**Processor**	**Memory**	**Bandwidth**	**Disk**
Processor	Cycles/s	Bytes	Mb/s	GBytes
Forwarding	Pkt/s	Number of rules	Mb/s	GBytes
Hybrid	Cycles/s + pkt/s	Bytes + number of rules	Mb/s	GBytes

3.4.2.3 Network Elements Description According to the Dimensions

The network elements present different profiles. These profiles are built according to the functions performed by the network element. SAVE defines three different element profiles. The first is the `forwarding element`, which describes a node that just forwards packets according to information on their tables. These nodes represent current routers and switches. The second profile is the `processing element`, which describes the network elements with the function of processing packets or requests. Hence, this profile models middle-boxes or servers. The third profile is a hybrid of the two others, called `hybrid element`. This profile models elements that forward packets, but also need processing power. Such an element could, for instance, authenticate messages hop-by-hop or perform advanced logging or intrusion detection systems. The three node categories can accommodate any possible node configuration in virtualized networks.

The element dimensions are differently described for each of these element models. For a `processing element`, the processor represents the number of cycles per second that the hired virtual machine can perform. Accordingly, the memory is set as the amount of physical memory that is set for the virtual machine to execute the user applications. Differently from the `processing element`, in a `forwarding element`, the processor represents the maximum volume of packets that the hired element can process. Since the memory required for packet forwarding is not a system bottleneck, then the memory for this kind of element can be set based only on the number of forwarding/routing rules required by the element. Because the hybrid node performs both roles of forwarding and processing, a network operator must specify the memory and the processor in both ways for this kind of node. Table 3.1 resumes the characteristics set for each element and Figure 3.4 describes in more details how the defined dimensions are correlated.

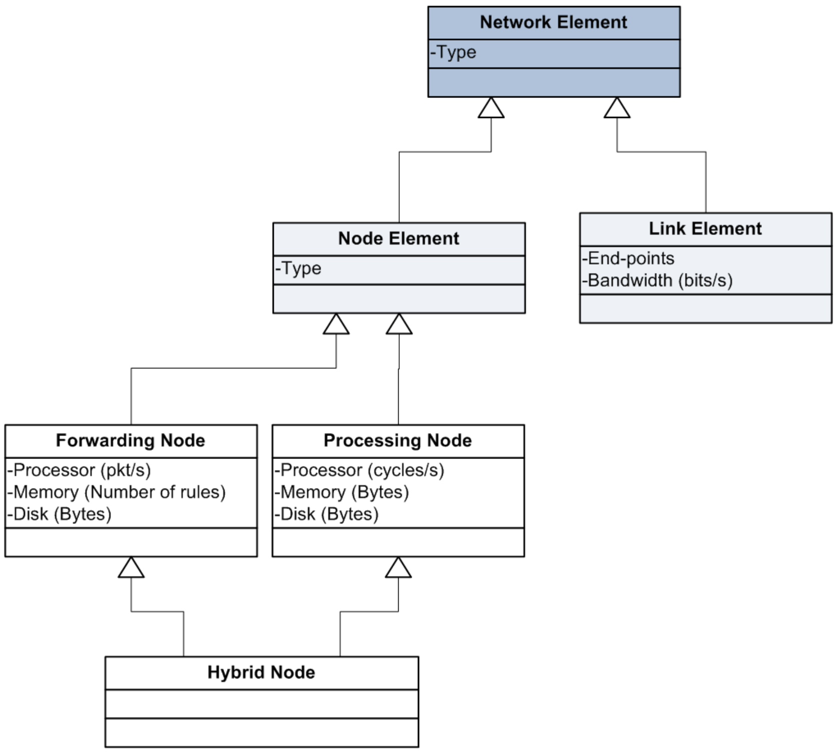


Figure 3.4: Diagram correlating the dimensions and types of network elements.

3.5 SAVE Control and Management Primitives

3.5.1 Primitives of Node Elements

The node elements are operated through different primitives. The basic primitives that control nodes are the instantiate and delete primitives. The instantiation of a node involves the selection of a virtual node disk image, the physical location of the node and the configuration of its parameters such as processor utilization, memory, bandwidth, etc. The deletion of nodes comprises the deletion of the node. The node is monitored through different daemons and the monitor primitive is used to retrieve information about the node such as processor usage, memory usage, associated links and their capabilities. Infrastructure operators use the modify primitive to migrate virtual elements among different physical nodes.

For each type of node element, different modify primitives are applied.

In the `processing elements`, the processor dimension can be modified in terms of the number of cycles/s allocated to the element and the memory can be modified in term of available megabytes. In `forwarding elements`, the processor dimension is controlled in terms of the number of packets that are forwarded per second. In these elements, the memory is managed in terms of the size of the rule tables and the operator can erase the table, insert, remove and modify table rules. `Hybrid elements` can be managed with the conjunction of management primitives of the processing and forwarding elements.

The primitives of node elements are summarized in Table 3.2.

Table 3.2: Primitives of node elements in SAVE.

Primitives of Node Elements
Instantiate node
Get node
Modify node
Delete node
Monitor node
Migrate node
Get processor
Set processor
Monitor processor
Get memory
Set memory
Monitor memory
Get memory_rule
Insert memory_rule
Modify memory_rule
Delete memory_rule
Monitor memory_rule
Get disk
Set disk
Monitor disk

3.5.2 Primitives of Link Elements

Link elements can be created and deleted. During the creation of the link, it must be specified the end points of the link, the bandwidth allocated to the link, queues and policies. SAVE allows operators to verify statistics of the links through monitor primitives. The monitor primitives for links allow the visualization of link statistics such as number of packets transmitted and packets lost and the modify primitive allows operators to change the allocated bandwidth, different queues and policies. It is also possible to insert links between end points and to remove links through the create and delete primitives. Table 3.3 shows a resume of the link element primitives provided by SAVE.

Table 3.3: Primitives of link elements in SAVE.

Primitives of Link Elements
Instantiate link
Get link
Modify link
Delete link
Monitor link
Migrate link
Set bandwidth
Get bandwidth
Monitor bandwidth
Set delay
Get delay
Monitor delay

3.5.3 Primitives of Interconnection

Given a set of nodes and link elements, there are primitives that allow operators to manage all elements simultaneously. The monitor topology primitive, for instance, lets operators to visualize all of their node and link elements. Another primitive concerning nodes and links is the migration, which is performed by the infrastructure manage. In the migration, the logical topology is remapped over the physical devices in order to achieve a specific objective. Table 3.4 shows the primitives of interconnection, which are correlated to the network traffic and the network topology.

Table 3.4: Primitives of interconnection in SAVE.

Primitives of interconnection
Instantiate topology
Get topology
Modify topology
Delete topology
Monitor topology
Migrate topology
Map topology
Set traffic_definition
Get traffic_definition
Modify traffic_definition
Delete traffic_definition

3.6 Mapping the High-level Primitives in the virtualization platforms

Different virtualization platforms have different ways to define and control a virtual network. A virtual network created by linking virtual machines or virtual containers works just as a physical network. Therefore, this kind of virtual network can be controlled just as it is done in a physical network. In this case, an administrator could use the standard distributed network control algorithms or project new mechanisms to be run by the nodes.

When different virtualization platforms are in use, such as the one proposed in OpenFlow, the network elements work in a predefined way and their behavior cannot be changed. Moreover, the network control must be centralized. Hence, mapping a distributed control mechanism in this kind of network is not trivial, especially because the nodes do not run different algorithms than the ones specified in the OpenFlow standard.

SAVE is intended to work independent of the virtualization platform used in the physical substrate. Hence, all the dimensions described in Section 3.4.2 and the primitives defined in Section 3.5 must be mapped in the different virtualization platforms. Indeed, the interconnection dimensions are platform independent, but the element dimensions are correlated to the virtualization platform characteristics and the physical-to-virtual mapping must be carefully designed.

3.6.1 Mapping virtual nodes and the corresponding primitives

A virtual node is not equally defined in all network virtualization models. Hence, mapping a node and its primitives requires specific knowledge about the underlying virtualization platform.

As described before, virtualization models such as OpenFlow does not allow network nodes to be modified. Instead, these nodes must follow the rules predefined in the standard. The network operator can only program the network controller, a central entity with access to all OpenFlow switches. When using machine or container virtualization, the network nodes can be programmed by the network administrator, which may perform a decentralized network control. Moreover, the nodes have processing power and can be used not only for forwarding packets, but also as a middle box.

Because of the differences between virtualization platforms, it is not possible to define a single model for network nodes. That is the reason why we defined processor, forwarding, and hybrid nodes. If the virtualization platform follows a model such as the proposed by OpenFlow, than we restrict the virtual nodes only to forwarding nodes. If a virtualization platform based on containers or virtual machines is in use, then we can create all the three kinds of nodes. If these conditions hold, we can provide a correct dimension mapping using SAVE. Nevertheless, the primitives mapping challenge still persist. Primitives such as create node, change resource allocation parameters, etc., are easily mapped, but configuring memory is a problem. Memory operations in forwarding nodes correspond to configure the data plane. Hence, SAVE must define a common interface to configure the data plane, which is to define forwarding and filtering rules, in a way that must be independent of a centralized or distributed network control.

3.6.1.1 Virtual node memory mapping

The infrastructure provider is responsible for managing the physical network, which is offered to the network operators. The virtualization platform is a choice of the infrastructure provider and cannot interfere in the virtual network control. Hence, the network operator must be able to define its control and management functions independent of the underlying virtualization platform. Indeed, SAVE assures that the network operator does not need to know the underlying virtualization platform for controlling its own network.

SAVE defines primitives for configuring the virtual network data plane that are independent of a central or distributed control plane in the physical or in the virtual network. Indeed, SAVE is based on the plane separation

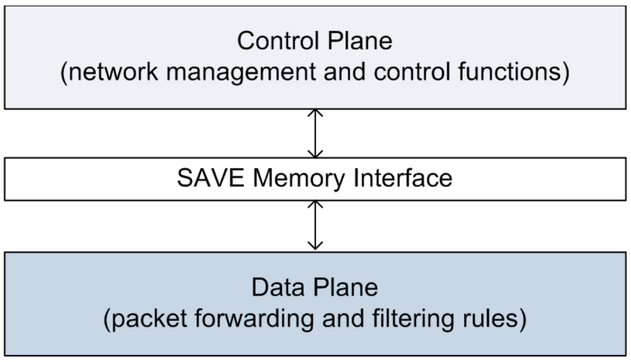

Control Plane
(network management and control functions)
SAVE Memory Interface
Data Plane
(packet forwarding and filtering rules)

Figure 3.5: Model for configuring the forwarding node memory using SAVE.

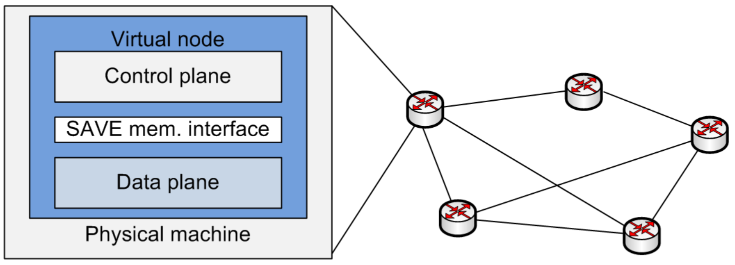


Figure 3.6: Virtual machine/container virtualization model using decentralized operator control.

paradigm. Hence, control plane and data plane do not need to run on the same device. SAVE inserts a control layer between data and control plane, as shown in Figure 3.5, in order to guarantee that the data plane can be configured independent of the network control chosen by the virtual network operator and by the infrastructure provider.

Figures 3.6, 3.7, 3.8, and 3.9 show how the data plane is configured, assuming the two network virtualization models. Figure 3.6 shows the classical network control, which is distributed and both the control and the data plane are on the same device. In this case, the SAVE memory layer intercepts the control plane requests to the data plane and convert them into the appropriate requests, according to the real data plane defined by the infrastructure provider.

Figure 3.7 shows the same model, but assuming that the network operator uses a centralized control. In this case, the centralized operator controller must configure the data planes using the SAVE memory interface. The physical routers of the infrastructure provider are able to receive these requests

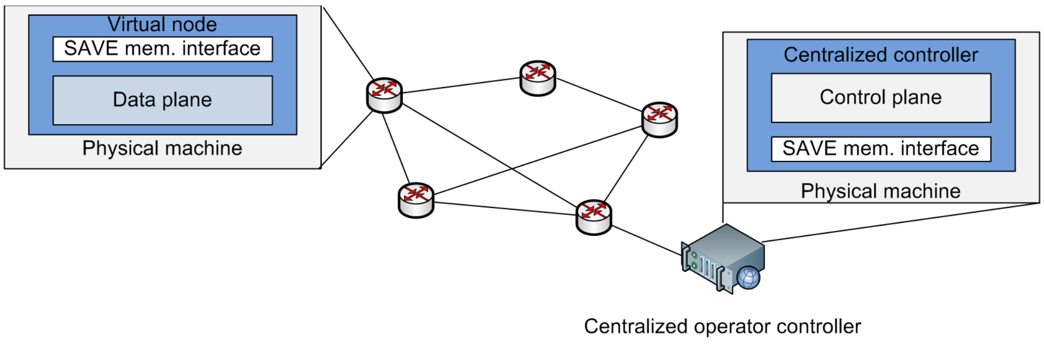


Figure 3.7: Virtual machine/container virtualization model using centralized operator control.

and translate them to the proper technology, allowing a correct configuration of the data plane of each virtual node.

Figures 3.8 and 3.9 shows the use of SAVE with the OpenFlow model, assuming the use of a distributed and a centralized control by the network operator, respectively. In the distributed operator control, SAVE maps the distributed control into a centralized control through the use of emulation, as suggested by Nascimento et al. [26]. Hence, the centralized infrastructure controller emulates a network for the network operator, which uses the SAVE memory interface to configure the emulated data plane. The centralized infrastructure controller intercepts the requests and uses them to configure the real data plane through the OpenFlow protocol.

In Figure 3.9, both the operator and the infrastructure administrator use a centralized network control. In this case, the centralized operator controller generates requests to configure the data plane using SAVE and these requests are intercepted by centralized infrastructure controller, which translate them into the OpenFlow protocol using the SAVE module.

Therefore, SAVE provides an interface to intermediate requests to configure the data plane that is compatible with different technologies in the virtual and in the physical layers. The key idea is that the network operator does not need to know how the infrastructure provider organizes and configures its devices, as soon as the virtual network data plane can be configured using the SAVE interface.

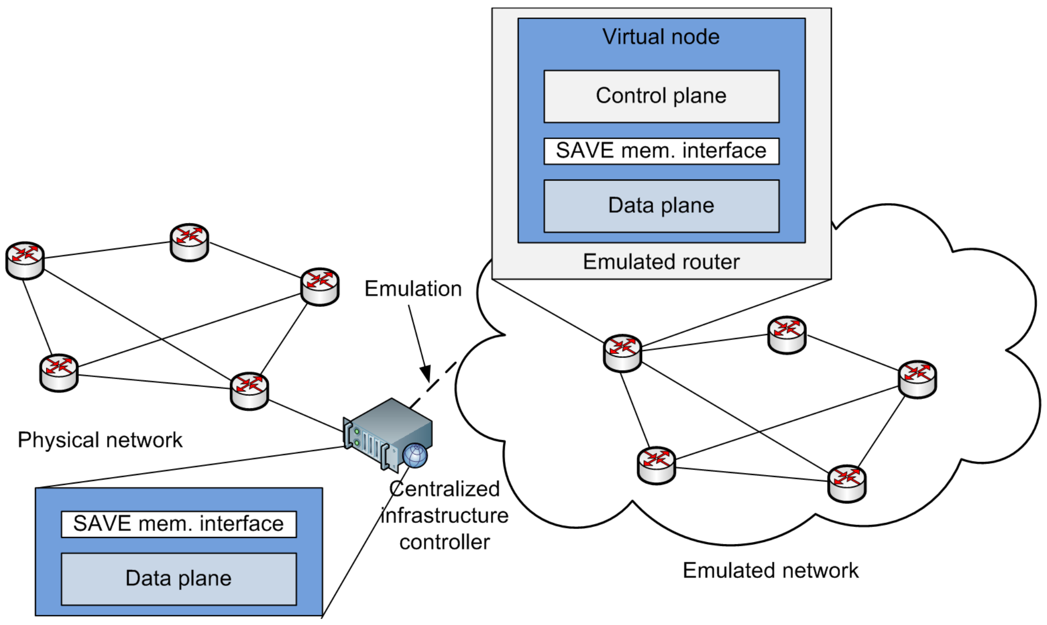


Figure 3.8: Centralized virtualization model using decentralized operator control.

3.6.2 Mapping virtual links and the corresponding primitives

Virtual links can be defined independent of virtualization platform. The link is an abstraction that does not change from physical to virtual substrates and is defined as a connection between two end points, as shown in Figure 3.10(a). The link can also have specific characteristics, such as bandwidth and delay, but these characteristics are equally defined in both physical and virtual networks. Therefore, mapping the dimensions of the link element is trivial.

Mapping virtual link primitives is not as easy as mapping the link dimensions, but it theoretically is platform independent. In both network virtualization models, a virtual link can be mapped in one or more physical links, just as described in Figures 3.10(b) and 3.10(c). Designing a link primitive that deals with virtual links mapped over more than one physical link is more complex than restricting the virtual network to the one-to-one link mapping, but it provides much more flexibility when mapping the virtual networks over the physical networks.

Hence, in terms of modeling, mapping a virtual link in an OpenFlow network or in a Xen network is just the same. It is important noticing that

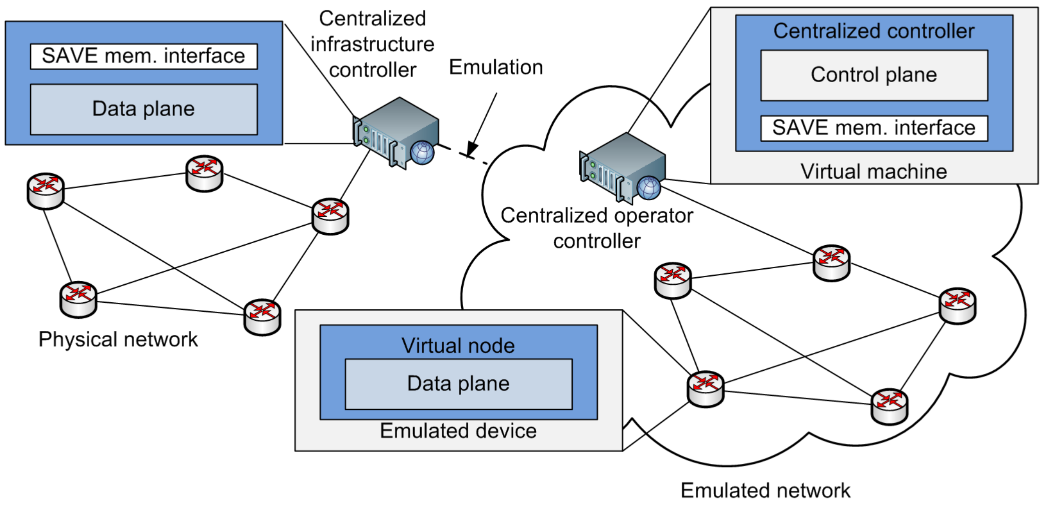


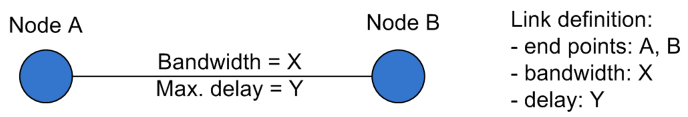
Figure 3.9: Centralized virtualization model using centralized operator control.

practical steps for migrating a virtual link, for instance, are different in the two platforms, because the control primitives are different. The virtual link migration concept, however, is just the same.

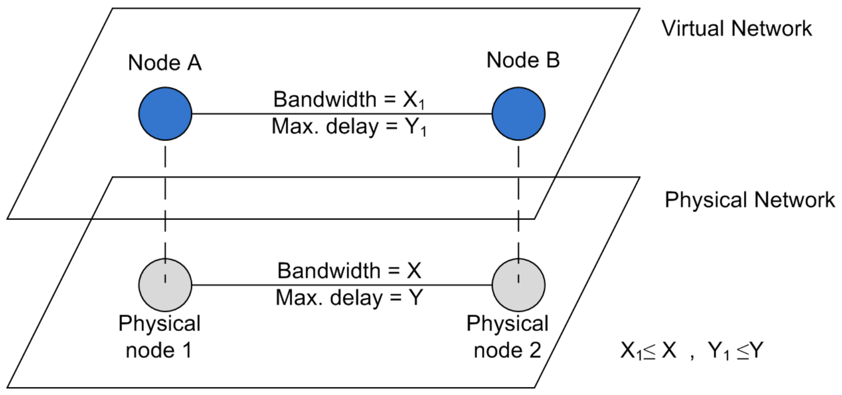
3.7 Prototyping the map of dimensions and high-level primitives in Xen and OpenFlow networks

In order to validate our proposal, we implemented a SAVE prototype, which works with Xen and OpenFlow virtualization platforms. These two platforms were selected because of their disparity in terms of network virtualization and management.

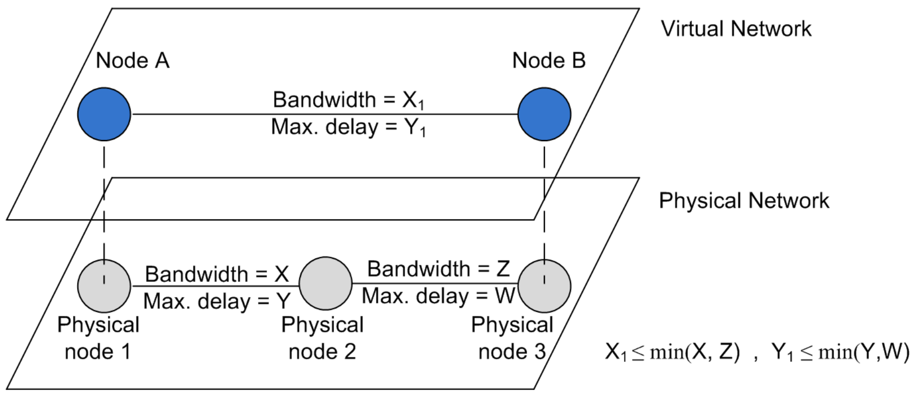
In this Section, we explain how to map dimensions and their primitives in each of the two platforms, which means building the `virtual substrate mapping` blocks for Xen and OpenFlow. We also develop experiments to generate mapping parameters and then we validate our mapping schemes through conformity tests.



(a) Defining a virtual link.



(b) One-to-one link mapping.



(c) One-to-many link mapping.

Figure 3.10: Mapping virtual links in physical links, which is platform independent.

3.7.1 Test Environment

We show now an example of how to use SAVE as a common interface for network virtualization for both Xen and OpenFlow platforms. We use a testbed composed of three machines, as described in Figure 3.11, to show the use of SAVE in a real scenario. The Traffic Generator machine (TG) sends

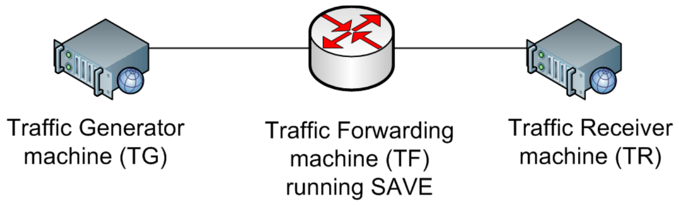


Figure 3.11: Test environment used for testing SAVE over different virtualization platforms.

packets to the Traffic Receiver machine (TR), and the packets are forwarded through the Traffic Forwarder machine (TF). TF is an HP Proliant DL380 G5 server equipped with two Intel Xeon E5440 (2.83GHz) processors and 10GB of RAM. Each processor has four cores. TF uses the two network interfaces of a PCI-Express x4 Intel Gigabit ET Dual Port Server Adapter. In the Xen experiments, TF executes Xen virtualization platform version 4.0 over Debian Linux operational system with kernel version 2.6.26. TG and TR are general-purpose machines equipped with Intel DP55KG motherboards and Intel Core I7 860 (2.80GHz) processors. TG and TR are directly connected to TF via their on-board Intel PRO/1000 PCI-Express network interface.

In the experiment scenario, TF represents an infrastructure provider that can offer virtual elements to clients. TG and TR represent end users that exchange packets between them. We used the Linux Kernel Packet Generator to generate packet flows. In order to create the mapping functions for Xen and to evaluate our mapping algorithms, we created a `hybrid element` (a node that forwards packets and perform extra processing activities despite of the forwarding activities) in TF. The node forwards packets of different sizes and flows with different packet rates. These results are used to calibrate the mapping algorithms. After that, we tested if the SAVE algorithms and the calibration were able to fulfill the resource requirements of the `hybrid element` for different client requirements. In our prototype, OpenFlow nodes work together with FlowVisor [27], which possesses native control to the interconnect dimensions, easing the control of them.

3.7.2 Processor mapping in Xen and OpenFlow

In SAVE, depending on the category of the node, network operators define the amount of CPU cycles or the packet rate to be sustained. We assume the existence of control mechanisms that ensure the provision of the defined amount of resources [28, 10, 29, 30, 31, 32]. Hence, SAVE does not treat isolation issues in virtualized environments. SAVE objective is to map the high–level parameters into parameters that make sense in each virtualization platform. Pure OpenFlow networks only define forwarding elements. In order to include processing elements, we must associate another virtualization technique which provides processing virtual environments, such as Xen and OpenVZ.

Defining the packet rate of the forwarding element is simple in OpenFlow networks, because this platform is oriented for virtual networking. Hence, FlowVisor definition [14] already presents a CPU definition based on the packet/s rate. Xen supports the creation of forwarding, processing, and hybrid elements. Nevertheless, the Xen platform demands a mapping between the SAVE parameters and the hypervisor scheduler parameters, in order to provide the proper CPU resource reservation.

Xen presents scheduler parameters to control the CPU sharing among virtual elements, which are the weigh and the cap. Weight represents a relative cycle division among virtual elements. For instance, a virtual element with weight 256 must receive twice the cycles received by a virtual element with weight 128. Besides that, the cap parameter defines the percentage of the maximum amount of CPU cycles received by the virtual element. In order to map SAVE parameters into proper Xen parameters, our Xen virtual substrate mapping uses Algorithm 1, which is based on the adjust of the CAP value and assumes that all virtual machines have equal weight. SAVE receives the required CPU cycles and the required packet rate and outputs the cap value for the virtual node.

In the algorithm, the `PckRateToCap` function uses a mapping table, which maps cap values that satisfies different packet rates. The mapping table for Xen is generated according to our tests seen in Figure 3.12. In the test, we measure the forwarding rates of network elements according to the input packet rate and the cap value in a Xen virtual element. We created different packet flows with 64 byte packets and measured the associated forwarding rate when cap varies. By this graph, we observe that there are minimal cap values to guarantee that all received packets will be forwarded. Based on this experiment, we mount a table with approximate values of CAP to guarantee a certain forwarding rate. This table is used to configure the CPU for forwarding nodes.

The processing nodes receive as CAP the percentage calculated based on the cycles/s rate indicated in SAVE input for CPU. The hybrid node is a composition of a processing and a forwarding node. Therefore, it receives as input parameter the CPU usage for processing and a packet rate. SAVE calculates the corresponding CAP for each kind of processing and sums them. This value is used as CAP for the virtual machine instantiated to the hybrid

Algorithm 1 CPU dimension mapping in Xen.

```
Input: $requiredCpuCycles, requiredPacketRate$
Output: $xenCapValue$
  $cpuMaxCycle := getMaxCpuCycle(physicalCore)$
  $cap = 0$
  if $nodeType == processor$ then
    $cap := 100 * requiredCpuCycles/cpuMaxCycle$
  end if
  if $nodeType == forwarding$ then
    $rateCap := pckRateToCap(requiredPacketRate)$
    $cap := rateCap$
  end if
  if $nodeType == hybrid$ then
    $rateCap := pckRateToCap(requiredPacketRate)$
    $cap := rateCap + (100 * requiredCpuCycles/cpuMaxCycle)$
  end if
  if $capToCycle(cap) <= cpuMaxCycle$ then
    $xenCapValue := cap$
  else
    $CannotAllocateResources$
  end if
```

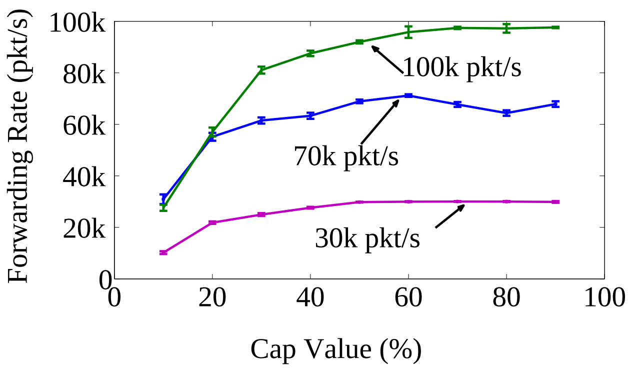


Figure 3.12: Forwarding rate as function of cap value for fixed input rates.

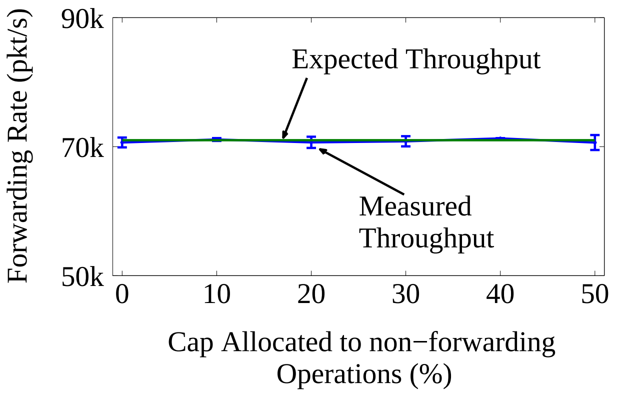
node.

To prove that this processing mapping for hybrid nodes works, we performed a second test. We assumed that a virtual network instantiates a hybrid node that requires a packet processing power correspondent to 70 kpkt/s and a processing power corresponding to x% of the CPU machine capacity. We varied x and verified if the hybrid node requirements were respected. We used the Linux tool ‘stress’ to generate the processing requirements according to x. Figure 3.13 shows the results.

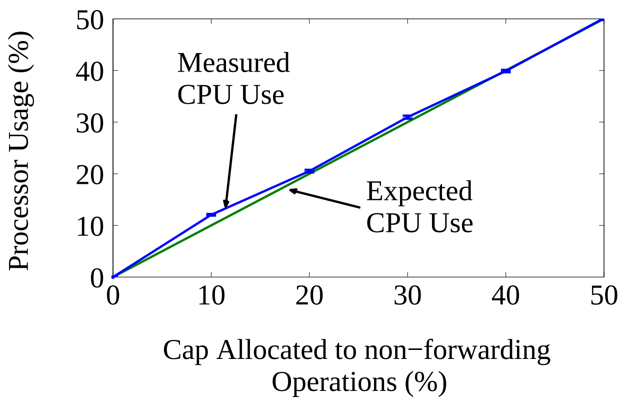
Figure 3.13(a) shows that independent of the amount of processing power requested by the hybrid node, the forwarding rate was respected, which means that the proposed map provides the correct CPU power for forwarding packets. Figure 3.13(b) shows the processor usage with the stress application in the same scenario. By this graph, the stress application received the agreed processing power for all x values, which means that our map also respect the provision of processing power for other applications rather than forwarding packets. Hence, the hybrid map works as expected and we can see that even if extra processor cycles are allocated to a virtual element, it keeps its forwarding rate and at the same time uses its extra processor cycles to execute extra processing activities.

3.7.3 Memory Dimension Mapping in Xen and OpenFlow

The memory dimension involves the allocation of memory to nodes. The memory is described according to the kind of node in SAVE. Indeed, for a `processing node`, it is important to define the amount of memory available



(a) Forwarding rate according to the CAP allocated to non-forwarding operations.



(b) CPU usage by the stress application according to the CAP allocated to non-forwarding operations.

Figure 3.13: Analysis of hybrid nodes processing map assuming the use of different processing requirements and a fixed forwarding rate of 70 kpkt/s.

for processing, but for `forwarding nodes`, the number of forwarding rules and the capacity for forwarding packets are the most important metrics. The virtual network operator is not able to map a number of forwarding rules to an amount of memory, because it depends on the underlying infrastructure. Hence, SAVE is responsible for doing this map.

In this Section, we show how to map the memory dimension in Xen and in OpenFlow. Therefore, we show how to specify and map the memory in `processing` and `hybrid nodes` for Xen and in `forwarding nodes` for Xen and OpenFlow.

We use our memory mapping algorithm, described in Algorithm 2, to allocate memory to Xen nodes. SAVE allocates to `processing elements` fixed amounts of memory in megabytes. The memory amount is a virtual machine parameter in Xen, specified before instantiating the virtual machine. For this reason, this is a simple map. For allocating memory to `forwarding nodes` in Xen, we must use a function, which maps the maximum number of forwarding rules allowed and convert it to memory space. This memory space must be enough to hold forwarding rules and the forwarding element operating system (bootSpace).

Algorithm 2 Memory dimension mapping in Xen.

```
Input: $requiredMemorySpace, ForwardingRules$
Output: $xenMemorySize$
  $maxMemorySize = getMaxMem(physicalMachine)$
  $requiredMemory := 0$
  if $nodeType == processor$ then
    $requiredMemory := requiredMemorySpace + bootSpace$
  end if
  if $nodeType == forwarding$ then
    $rulesSpace := rulesToSpace(ForwardingRules)$
    $requiredMemory := rulesSpace + bootSpace$
  end if
  if $nodeType == hybrid$ then
    $rulesSpace := rulesToSpace(ForwardingRules)$
    $requiredMemory := requiredMemorySpace + bootSpace + rulesSpace$
  end if
  if $requiredMemory <= maxMemorySize$ then
    $xenMemorySize := requiredMemory$
  else
    $CannotAllocateResources$
  end if
```

As shown in the Algorithm, the amount of memory to `hybrid nodes` is a composition of the memory to processing nodes and the memory required by the forwarding rules.

The memory dimension map in OpenFlow networks reefers only to forwarding nodes and is very simple. FlowVisor, which configures virtual networks in OpenFlow, accepts as input for memory control the number of forwarding rules of each virtual network. Hence, SAVE configures FlowVisor with the received value, using the `Virtual Substrate Mapping` module of OpenFlow.

Therefore, SAVE provides the proper functions for mapping element dimensions into different virtualization models.

3.7.4 Implementing the control and management primitive mapping

We show now more details about the mapping of the control primitives, in which we describe the initial prototype developed for SAVE. We focus in the development of four main modules of SAVE architecture:

- HTTP Server & Controller - This server receives service requests, validates parameters, and verifies whether the requested service is supported in the requested virtualization platform. If all requirements are supported, then this module forwards the packet to the Virtualized Substrate Handlers (VSH). In this initial implementation, which works as proof of concept, we opted to implement both the HTTP server and the controller as a single module for simplicity. We restricted the available functions provided by this module to the essential function set of the HTTP Server and the Controller.

- Mapping Knowledge Base - This is a data base containing indications of the current supported network virtualization platforms.

- Virtualized Substrate Handlers - The VSHs map the service calls of SAVE in the prototype services in the specified virtualization technology. Hence, we developed a VSH for Xen and a VSH for OpenFlow. The VSH sends a call to the control client in the virtualization platform.

- Control client in the virtualization platform - SAVE maps generic request into platform specific requests. We used VNEXT [33, 34] and OMNI [35, 36], developed in the previous work packages, as clients for Xen and OpenFlow platforms, respectively.

The scheme of our prototype is shown in Figure 3.14.

It is important mentioning that it is very easy to extend SAVE to other virtualization platforms, as soon as there is a client for that platform that performs virtual network management functions. That is the main reason for creating the VSH as separated modules. Also, with this architecture, we can extend and modify a VSH without interfering with the management of other virtualization platforms. A new virtualization platform is integrated by inserting a new entry in the Mapping Knowledge Base and by adding the corresponding new VSH.

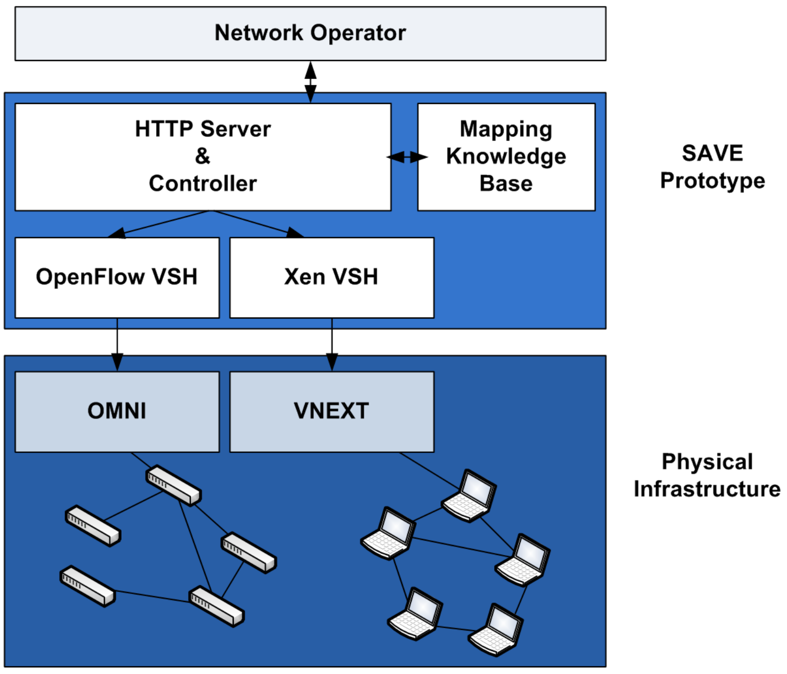


Figure 3.14: Developed prototype of SAVE, which works with Xen and OpenFlow platforms.

When the HTTP Server & Controller module receives a request, it searches for the requested technology in the Mapping Knowledge Base to obtain access to the corresponding VSH. After that, it forwards the request to the VSH.

The VSHs are written as python modules to simplify expansions and are placed on the same directory. The directory of the new module must follow the model <technology>_client, where <technology> is the name of the new virtualization platform. For instance, if we create a virtualization platform called "HorizonModel", then the directory of the new module would be called HorizonModel_client and this new directory is placed under the SAVE root directory. Hence, if our server supports HorizonModel, OpenFlow, and Xen, then we would have the following organization in the directories:

```
virtualnetworkserver/HorizonModel_client/
virtualnetworkserver/OpenFlow_client/
virtualnetworkserver/Xen_client/
```

The corresponding VSH instantiates an object of the class Client to deal with the control client of the virtualization platform. In this implementation, this object receives the parameters that identify the control client server, which means the identification of the server and the TCP/UDP port. It is not part of SAVE scope to develop the control client. In case of a distributed network control, the identification field may contain an identification of the set of nodes instead of the identification of a single server.

A simple code for the class Client run by the server is in Algorithm 3.

Algorithm 3 A simple code for the class Client run by the SAVE server.

```
class Client():
        def _init_(self, server = None, port = None):
                self.server = server
                self.port = port
```

This class is stored in a file called client_<technology>.py inside the directory <technology>_client.

3.7.4.1 Implementing the VSHs

The Virtualized Substrate Handlers maps the services of SAVE in the services of the virtualization platform, following the standards defined by SAVE. The VSHs respect these rules:

- the VSH is written as a Python class;
- the class name is Handler;
- the constructor receives as parameter an instance of the class Client;
- the VSH must import the class Client and the module httplib;
- the VSH must present a service for each service offered by SAVE, and both services must have the same name;
- all the parameters of a service are passed as a single parameter organized as a dictionary with the tuples {'parameter':'value'}; and
- all functions that perform a service must return a tuple containing two elements, the error code of the httplib module and the XML message with the service reply.

An example of part of a VSH is presented in Algorithms 4 and 5.

Algorithm 4 An example of code for a VSH concerning link primitives in SAVE.

```
from technology_ client import Client
import httplib
class Handler():
        def _init_(self, tc = None):
                self.tc = None
                self.setTechnologyClient(tc)
        def setTechnologyClient(tc):
                if isinstance(tc, Client):
                        self.tc = tc
        def getVirtualNetworkBandwidth(self):
                message = "<get_virtual_network_bandwidth>%s</get_virtual
_network_bandwidth>"
                return httplib.OK, message
        def setVirtualNetworkBandwidth(self, param):
                message = "<set_virtual_network_bandwidth>%s</set_virtual
_network_bandwidth>"
                return httplib.OK, message
        def monitorVirtualNetworkBandwidth(self, param):
                message = "<monitor_virtual_network_bandwidth>%s</monitor
_virtual_network_bandwidth>"
                return httplib.OK, message
        def getVirtualNetworkDelay(self):
                message = "<get_virtual_network_delay>%s</get_virtual_
network_delay>"
                return httplib.OK, message
        def setVirtualNetworkDelay(self, param):
                message = "<set_virtual_network_delay>%s</set_virtual_ net-
work_delay>"
                return httplib.OK, message
```

Algorithm 5 An example of code for a VSH concerning link primitives in SAVE (continuation).

```
    def monitorVirtualNetworkDelay(self, param):
        message = "<monitor_virtual_network_delay>%s</monitor
_virtual_network_delay>"
        return httplib.OK, message
    def instantiateVirtualNetworkLink(self):
        message = "<instantiate_virtual_network_link>%s</instantiate
_virtual_network_link>"
        return httplib.OK, message
    def getVirtualNetworkLink(self, param):
        message = "<get_virtual_network_link>%s</get_virtual_network
_link>"
        return httplib.OK, message
    def modifyVirtualNetworklink(self, param):
        message = "<modify_virtual_network_link>%s</modify_virtual
_network_link>"
        return httplib.OK, message
    def deleteVirtualNetworklink(self, param):
        message = "<delete_virtual_network_link>%s</delete_virtual
_network_link>"
        return httplib.OK, message
    def monitorVirtualNetworklink(self, param):
        message = "<monitor_virtual_network_link>%s</monitor_
virtual_network_link>"
        return httplib.OK, message
    def migrateVirtualNetworklink(self, param):
        message = "<migrate_virtual_network_link>%s</migrate_
virtual_network_link>"
        return httplib.OK, message
```

3.7.4.2 Implementing SAVE general functions

SAVE requires functions that are not correlated with the virtualization platform, but with its own management. We provide a summarized list:

- getServiceList - Since an initial implementation may not provide all the listed services, we created a service to list all the supported services. This service is called getServicesList and returns an xml with the list of the available services without the required parameters.

- getSanityTest - This service tests the connectivity between SAVE and the physical infrastructure and between SAVE and the network operator.

3.7.5 Using SAVE

SAVE is available online in the Horizon Project page. The corresponding code and manuals are available for installing and using the developed system. More information about the developed functions is found in the web site.

Chapter 4

Piloting plane control with multi-agent systems

We propose on the Horizon Project to use autonomic networks to deal with the virtual network management problem. Autonomic networks represent a specific topic in the area of autonomic computing, a term coined by IBM, intended to deal with complexity by enabling systems to self-manage. Today, it is also advocated the approach of pluralism of architectures for the future Internet over the one-size-fits-all TCP/IP [37]. This new approach defines that network providers should be divided in service and infrastructure providers [38] and proposes the use of virtualization [39]. Users request network services from the service providers, which instantiate virtual networks over the substrate provided by the infrastructure providers. Each virtual network can have its own protocols and configurations, in accordance with the objectives of the service running on it, and must have isolation, i.e., the operation of virtual networks does not cause interference between them, although they are on the same infrastructure.

4.1 Background

4.1.1 Multi agent Systems

Multi-agent systems [40] are composed of intelligent entities, called agents, which have the capabilities needed to make the network autonomic. As shown in [40]: (1) they are able to communicate, (2) possess their own resources, (3) perceive their environment, (4) have a partial representation of their environment, (5) have a behavior which aims at realizing their goals.

Thanks to such properties, multi agent systems can constitute a good tool to provide the autonomic scheme by guaranteeing the different characteristics which seem necessary to reach an autonomic behavior. In the following, we describe in more detail multi-agent systems characteristics:

- Decentralization: The multi-agent approach is decentralized by definition and this decentralization aims at overcoming the incapacity of the classic Artificial Intelligence to operate in the current systems that are more and more distributed and decentralized. No agent possesses a global vision of the system and the decisions are taken in a totally decentralized way;

- Reactivity: One of the basic attributes of an agent is to be situated (situatedness, [41]). That is, an agent is a part of an environment and it reacts according to what it perceives of this environment. The reactivity characteristic is very important in a context of highly dynamic networks, in which the decisions have to suit current conditions.;

- Proactivity: The agent is capable of setting goals and realizing them by executing plans, interacting with other agents, etc. In this case, the agent has more knowledge of its capabilities and on those of the other agents and is able to set up a strategy allowing it to evolve in its environment and to reach its objectives;

- Sociability: The multi-agent approach provides the ability to distribute the intelligence among different agents composing the system. This implies that an agent can handle some tasks individually but cannot do everything by itself. It needs to cooperate with the other agents and to rely on their help to get better results;

- Adaptability: In order to provide more flexibility, researchers are interested in using learning techniques (e. g., genetic algorithm [42], reinforcement learning [43], etc.) to face unexpected situations. If we return to the autonomic networks, using agents having learning capacity can be very beneficial and allows for a more effective adaptation to the evolutions of the networking domain.

4.1.2 Norms

According to [44] a norm has a general structure of a group of agents. Thus, a norm consists of four components: the addressees' agents, the action to be performed by them and, finally, the circumstances under which the action must be carried out [44, 45, 46]. Moreover, [44] classifies norms as one of two kinds: rules or (r-norms), and social norms or (s-norms).

Rules represent explicit agreements among agents, and are created by an authority. Rules are subdivided into two further classes as follows. Formal rules are those that include legal sanctions such as laws and regulations, and informal rules that are not in written form but communicated orally and include informal sanctions.

Social norms are norms accepted not through agreement but through mutual beliefs, and are also divided into two classes: conventions, which concern the whole society or social class and have social sanctions, such as approval or disapproval; and group-specific norms, which concern a group of agents in a society.

[44] also explains the conditions under which either rules or social norms ought to be fulfilled by the members of a group; these conditions cause a norm to be enforced and can be described, as follows: (i) promulgation condition refers to the fact that norms must be issued by an authority; (ii) accessibility condition states that all members of the group acquire the belief that they ought to comply with the norm; (iii) if many members of the group fulfill the norm, or at least are disposed to do so, it is said that the pervasiveness condition is satisfied; (iv) the motivational condition is met when at least some members sometimes fulfill the norm because they believe it is true and that they ought to do so; (v) the sanction condition refers to the existence of social pressure against members that deviate from the norm, and, finally, (vi) for a rule, the acceptance condition is the conjunction of the promulgation and accessibility conditions, whereas for a proper social norm to be accepted, only the accessibility condition is needed. Thus, contrary to rules, s-norms do not need to be issued by an authority, but they have to be recognized as norms for all the members in a group. In this work, we consider rules as mechanism to regulate the agent's behaviors.

4.1.3 Self-Organization

The approach of self-organizing systems has increased its relevance and is used to deal with complex domains. The use of this approach enables the development of decentralized systems that exhibit certain dynamicity and adaptability to couple with previously unknown perturbations [47]. According to principles of self-organization, each component of the autonomic system obtains and maintains only local information available in the environment, in a decentralized way and without any external control, being restricted only to local interactions. It is based on these interactions that the system exhibits it macroscopic behavior, which may be observed from a global point of view. The multi-agent system paradigm has been considered a promising solution for the building of self-organized systems [48]. According to [47] self-organization systems can be classified, as follows:

- Strong self-organizing systems: are those where there is no explicit control, whether internal or external.
- Weak self-organizing systems: are those where there is a re-organization through actions of center or planned internal control.

Additionally, the behaviors of a self-organization system can be characterized by the following properties (mandatory or optional):

- Absence of an explicit external control - This is a mandatory property that indicates that the system is autonomous, which defines change and that its organization is based exclusively on internal decisions and does not follow any external control to perform a (re-) organization. This property refers to the 'self' of self-organization.
- Decentralized control - A self-organizing system can work under decentralized control. In this case, there is no internal central authority or centralized information flow. In this way, the access to global information is limited by local interactions, which are governed by simple rules. This property is generally not mandatory, as we can see it in natural self-organizing systems, such as the bees.
- Dynamic Operation - This mandatory property is associated the evolution of the system. Considering that the organization evolves independently of any external control, this property implies in the self-organization process.

4.2 State of art

Telecommunications Network Management systems are a type of system that can be categorized as large, complex and unpredictable. Current research focuses on policy based management and autonomous systems, using a variety of languages and technologies. The three main Autonomic Network Management systems are ANEMA [49], FOCALE [50], and Pronto [51]. We describe, critique, and compare them with our piloting approach.

In ANEMA, the high-level objectives of the human administrators and the users are captured and expressed in terms of Utility Function policies through a set of mechanisms. The Goal policies describe the high-level management directives needed to guide the network to achieve the previous utility functions. Finally, the 'behavioral' policies describe the behaviors that should be followed by network equipment to react to changes in their context and to achieve the given ‘Goal’ policies.

To demonstrate the capacities of the ANEMA architecture, they explained how it should be instantiated in a multiservice IP network and how the proposed utility-based analytical models were used. The results confirmed that the proposed model allows specifying the optimal feasible state. However the result state is still not the optimal one. They also implemented a simulator of the system and perform a set of simulations based on several proposed scenarios.

One problem to be solved in ANEMA is how the autonomic routers diffuse the information to others routers on the environment. Another issue that is not so clear is how deep the coupling between the stock router and an autonomic one is. The ANEMA focuses on the self-configuration and a little on the self-optimization aspect, thus is not a complete autonomic solution, as our piloting approach. Neither they use biological inspired solutions.

On the other hand, FOCALE, Foundation Observation Comparison Action Learn rEason, propose the use of information and ontological modeling to capture knowledge relating to network capabilities, environmental constraints, and business goals and policies, together with reasoning and learning techniques, to enhance and evolve this knowledge. Also, to deliver full autonomic network management capabilities FOCALE introduce decentralized processes and algorithms into the network infrastructure modeled on various biological processes found in the nature world. As ANEMA, FOCALE uses policy-based network management system, incorporating translation/-code generation processes that automatically configure network elements in response to changing business goals and/or environment context.

FOCALE have as a base element an AME (Autonomic Management Element) which handles a managed resource, be it single device or network, which is the same idea of our piloting agent. Also, FOCALE is based on the MAPE control loop described by [52], and used by our solution, the Monitor, Analyze, Plan and Execute loop, but reduced to a maintenance control loop and an adjustment control loop.

Finally, Pronto specifies a Policy-based service definition language to describe services and the system model through service definitions. The language merely allows those services to be described by a network engineer, but it is responsibility of the management system to use the policies within the service definition to construct and manage individual services.

Policies can also be applied to a pluggable and automated management software component known as a Domain Expert. These components transform policies at a high level of abstraction into corresponding lower-level policies. A QoS Domain Expert instance with dynamic behavior will be used if congestion is detected, the Domain Expert will modify the low-level policies to reduce the Committed Information Rate (CIR) of each service.

Basically, the Pronto solution specifies a domain specific policy language, which defines desired parameters of each network device. There isn't an autonomic module or agent for each network element, but a virtual device, which controls the configuration of the associated network device. Here, the term virtual device is not the same as we use in our piloting network. The authors didn't describe the whole architecture, so we don't know if it is centralized or distributed, or how the policies are diffused to others devices to couple with unpredictable situation. They do not use agents or biological inspired solution.

4.3 A Self-Organizing and Normative Piloting System

In order to provide autonomy to the virtual networks, we developed a multi-agent self-organizing and normative piloting system. The idea is to adjust the network flow and routes in an autonomic way, without any explicit central control, maintaining the quality of service defined in the SLA and controlling the agent behaviors through norms. So, the virtual networks devices have a piloting agent responsible to capture and diffuse information among neighbors and act in the configuration and management of the router under a local perspective.

The piloting system operates mainly in the core network, i.e. the routers, as Figure 4.1 shows. Initially, agents are assigned to each router, and immediately retrieve information from the router they belong, like the routes they attend. After this step, the agent will be aware of the norms (i. e., SLA and QoS requirements) that need to couple and the normative regulation system can prohibit access to the network to those agents that violate the norms. Essentially, agents can play the following conducts: (Abiding) always abides by the norm; (Violating) may violate the norm; (Friendly) always consents to interact with others agents; and (Hiding) will avoid interact with those that violate norms.

To complete the information relevant to the piloting system, agents make contact with the others agents on their neighborhood using the behaviors defined in Section 4.3.1. The neighborhood in the piloting system is defined as the node (router) connected by a link, or just one hop. For example, in Figure 4.1, the Router #3 has as neighbors the Routers #2, #4 and #5. Thus, for each router in the neighborhood will be requested its routing table

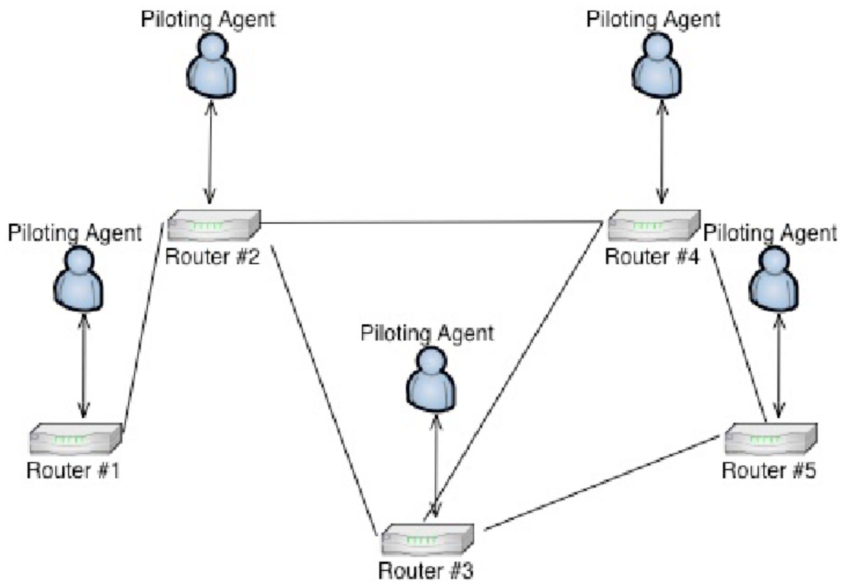


Figure 4.1: Neighborhood.

according to the routing table of the requester, its capacity to meet QoS, its current load and its average load. The latter information is requested randomly or just before a decision-making, i.e., after exceeding a threshold load. Being aware of the neighboring node ability will enable the agent to delegate or ask for routes in the inability to meet a particular request, or provide services in accordance with the requirements of quality requested.

4.3.1 Agent Behaviors

The piloting agent behaviors are:

- **Collect Behavior**: This behavior is responsible for collecting and storing data from the local and the neighborhood routers, the latter executed by the behavior Request Information Behavior.
- **Analyze Behavior**: This behavior analyzes the data collected and checks if they are in accordance with the quality policies required. It is also responsible for activating the decision behavior, described below.
- **Decision Behavior**: This behavior is composed of mechanisms and algorithms of decision making restricted to the data collected locally. It is possible to extend this approach to define different mechanisms for decision in accordance with the needs of the piloting system.

- **Response Information Behavior**: This behavior is responsible for serving the information request from the neighboring agents; routing tables, current and average load are sent. This behavior can be extended to address the need of other types of information in accordance with the piloting system.

- **Request Information Behavior**: Behavior responsible for requesting the local information necessary for analysis and decision making of the piloting system.

- **Create Router Behavior**: This behavior comes into play when after the decision process, the action to be taken is to create a new virtual router to meet the actual demand of the network. Thus, this behavior contacts the network simulation engine requesting the instantiation of a new virtual router.

- **Create Piloting Agent Behavior**: Complementing the previous behavior, this behavior is instructed to ask the agent platform the instantiation of a new piloting agent to control the new virtual router.

- **Delegate Route Behavior**: This behavior is responsible for delegating the adherence to a particular flow to a virtual router in the neighborhood. It can be activated either after creating a new virtual router, as the perception of a neighbor with load available to meet current demand.

- **Inform Route Behavior**: Behavior used to communicate to the neighboring router, which is a flow generator, which the route was modified to conform to the actual quality criteria, and therefore the router needs to update its routing table.

4.3.2 Acting on a QoS Failure or Malfunction

Essentially, this is a feature of the self-configuration and self-healing autonomic piloting system. When answering a particular request for data traffic, the router and therefore its pilot agent will know which conditions and QoS must be satisfied. So, monitoring the router performance, like quality requirements, which can be configured at runtime by the network administrator, initializes the agent actions. Thus, once the agent detects a non-fulfillment, or the inability to meet quality concerns by the router, it considers whether they have enough information from their neighbors to be able to make a decision. If the agent finds that the information is outdated and that the problem is yet just a trend, it will request updated information from its neighbors, however, always monitoring the current performance of the router. After obtaining these data, the agent uses their algorithms and behaviors for deciding on their actions to solve the current problem in a decentralized approach.

Thus, the agent analyzes the routing tables of its neighbors verifies if it also serves the route that currently requires higher quality and it also checks if the neighbor has available load to provide. In a positive case, the agent will delegate this data stream to the neighboring router, performing a piloting action at runtime. However, if no neighbor is able to meet such demand, the agent will instantiate a new virtual router, associate a new piloting agent and then delegate the flow to the new virtual router. After that, the agent must communicate with the other neighbor from where the flow is coming, to change the route.

In case of malfunction, the agent can replicate all the routes it serves to a new virtual router, and trigger an alarm for a human intervention to address the occurrence of the error, as this would be outside the scope of agent autonomy.

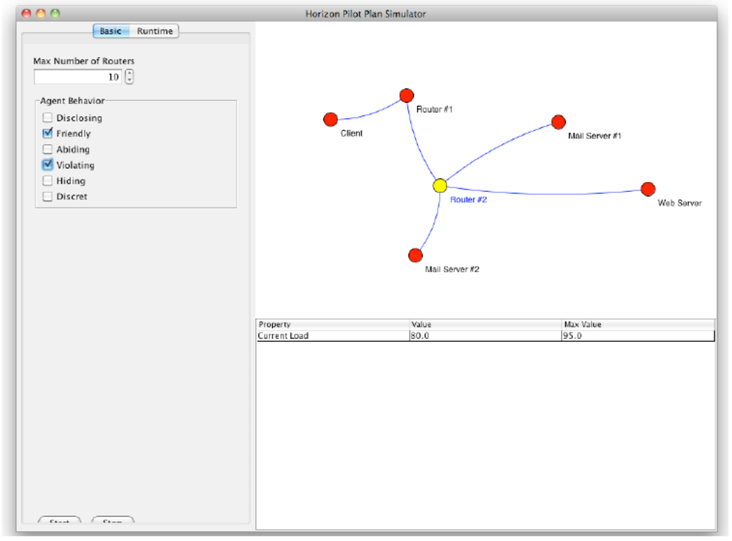
4.4 Piloting plane simulator

We have implemented a simulation environment for the scenario previous described. We also have implemented the solution presented in Section 4.3. See Figure 4.2 for a screenshot of the simulator. The aim of the simulation environment is to provide users with the functionality to experiment with agent-oriented pilot plans.

Following the architecture outlined in Section 4.4.1, each node represents a virtual device. Internally the simulator maintains three different types of data:

- the virtual devices and the current value of their observed properties;
- the routes; and
- the routers that compose the virtual network.

Each time the simulation clock is incremented, the virtualization environment is consulted, and the properties values of each device are updated. Therefore, each agent is responsible for managing its local data using the API explained in Section 4.4.2. Once the multi-agent system decided to instantiate or change the virtual network, it executes the solution outlined in



Max Number of Routers: 10

Agent Behavior
- [ ] Disclosing
- [✓] Friendly
- [ ] Abiding
- [✓] Violating
- [ ] Hiding
- [ ] Discret

Property	Value	Max Value
Current Load	80.0	95.0

Figure 4.2: Simulation Environment GUI

Section 4.3, and wait for the new set of virtual routers/routes. The difference between the new set of routers/routes and the older one is computed and used to instantiate a new virtual network using the API. In reality, the algorithm may behave in different ways. To illustrate these different behaviors, the simulator offers the ability to modify the different agent's behaviors (see Section 6.2) and simulation parameters, such as: amount of data and data transfer rate. For example, if an agent has a Violating behavior and there is a rule defining that "if the data transfer rate is below a threshold, the agent has to reject the requests to transfer data", such agent can violate such rule. Otherwise, if the agent has an Abiding behavior, it will fulfill the rule, i.e., it will not transfer data and a new virtual router must be defined.

4.4.1 Architecture Overview

The architecture of our proposed simulation environment is composed of three elements: (i) a user interface; (ii) a normative multi-agent system; and (iii) a virtualization environment, as shown in Figure 4.3. The user interface provides users with the functionality to visualize the network and control simulation parameters. The multi-agent environment implements the algorithms as presented in Section 4.3. Finally, the virtualization environment provides access to virtual devices and means to instantiate and re-instantiate virtual networks. The communication between the user interface and the simulation environment is performed via a standardized programming interface

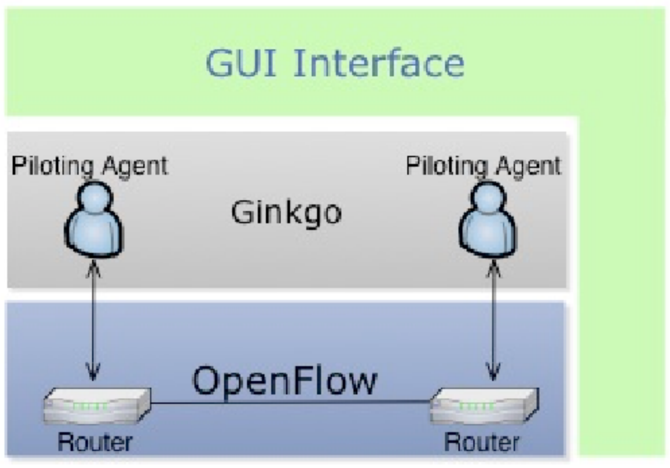


Figure 4.3: Simulator Architecture

(see Section 4.4.2). The advantage of the programming interface is to decouple the normative multi-agent system from the virtualization environment. Therefore, the simulation environment can be easily portable across multiple virtualization environments (e.g., Xen [53], OpenFlow [54]).

4.4.1.1 GUI

The graphical user interface allows users to experiment with the proposed pilot system. The user interface provides users with the functionality to visualize the network topology, devices and routes. In addition it allows users to control some simulation parameters like the: max number of virtual routes that can be instantiated in each simulation and the behaviors that agents may assume. The GUI was implemented in Java using the JUNG library to represent the network topology. JUNG [55] (Java Universal Network/Graph Framework) is a library that provides a common and extensible language for the modeling, analysis, and visualization of data that can be represented as a network. The JUNG architecture is designed to support a variety of representations of entities and their relations, such as directed and undirected graphs, graphs with parallel edges, and so on. It also provides a mechanism for annotating graphs, entities, and relations with metadata. This has facilitated the creation of the functionalities that examine the relations between devices as well as the properties attached to each device and relation.

4.4.1.2 Ginkgo

The agents were implemented using the Ginkgo platform [56], through the construction of several behaviors. Using those behaviors the agents can exchange information among neighbors, store, analyze and decide what to do to attend the imposed QoS requirement, for example. The Ginkgo Distributed Network is an agent platform based on autonomic networks. It has the building blocks for the development of a piloting system for computer networks. The framework allows the creation of lightweight and portable agents, which facilitates its implementation in heterogeneous environments: routers, switches, hosts, wired and wireless networks. The agents play the role of the autonomic manager of autonomic computing. With distributed managers near its managed elements, monitoring can be done locally. The platform also allows the formation of clusters of agents in neighborhoods. Neighbors exchange information and get a situated view of the network. Thus, besides the local environment, the agent is aware of other network places. This information is stored in the knowledge base that has an information model to facilitate communication between agents. Other data repository is the policy file, which contains rules of the application. In our pilot system, rules are interpreted as norms. The behaviors described in Section 4.3.1 are realized as Gingko behaviors. They feed the knowledge base, perceive and predict threatening events and perform changes on the managed virtual devices. In Gingko agents also may have a dynamic planner that, with information in the knowledge base and the rules in the policy file, changes parameters of the behaviors and controls the life cycle of the agent. This makes possible to develop the properties of self-configuration, self-healing, self-optimizing, and self-protection in the network, which promote the self-management.

4.4.1.3 OpenFlow

For the virtual network, the OpenFlow system was used. The OpenFlow provides an open protocol to program the flow table in different switches and routers. A network administrator can partition traffic into production and research flows. Researchers can control their own flows – by choosing the routes their packets follow and the processing they receive. In this way, researchers can try new routing protocols, security models, addressing schemes, and even alternatives to IP. On the same network, the production traffic is isolated and processed in the same way as today.

In order to connect the OpenFlow system and the Ginkgo platform we used the Beacon controller, which is a Java-based OpenFlow controller, built on an OSGI [57] framework, allowing OpenFlow applications built on the platform to be started, stopped, refreshed, installed at run-time, without disconnecting switches. Beacon has the following features that helped us on the development of the simulation environment:

- Cross-platform - Runs anywhere Java runs (including embedded devices, e. g., switches and routers);
- Dynamic - Code and resource bundles can be started, stopped, refreshed, installed at runtime, including dependent bundles, without disconnecting switches;
- Embedded J2EE Web server [58]- Jetty [59] is optionally embedded enabling a fully capable enterprise web server;
- Unit testing - Support for JUnit [60] unit testing;
- Maven [61] - Beacon can be built using Maven, and exported to Maven and P2 repositories.
- Performance - Beacon has been tested and shown to service 250,000 L2 switch Packet-In requests per second in single threaded mode on a 2.4 GHz Core 2 processor using 512MB of RAM. Widening the thread count to 3 increases performance to 340,000 Packet-Ins/s.

4.4.2 Programming Interface

The programming interface defines how to integrate the simulation environment with any virtualization environment. This section provides a comprehensive description of the proposed programming interface. The aim of the interface is to allow the simulation environment to perform tasks such as: (i) obtaining the available physical devices and routes; (ii) getting the instantiated virtual network; (iii) observing values of devices properties; and (iv) reconfiguring the virtual network. Figure 4.4 provides an overview of the concepts that compose the interface.

The Device concept represents the physical network devices, such as routers. Each device has a unique ID, a name and a set of Property elements. A property is a pair of name and value uniquely identified by an ID. The routing tables are represented by the Route concept. A route maintains the linked devices and is identified by an ID. A virtual network is a sub set of the set of routes that compose the physical network.

In order to manage the virtual network, the programming interface provides a function that supports the piloting system to manage the virtual network by adding and removing routes and devices, accordingly. Therefore

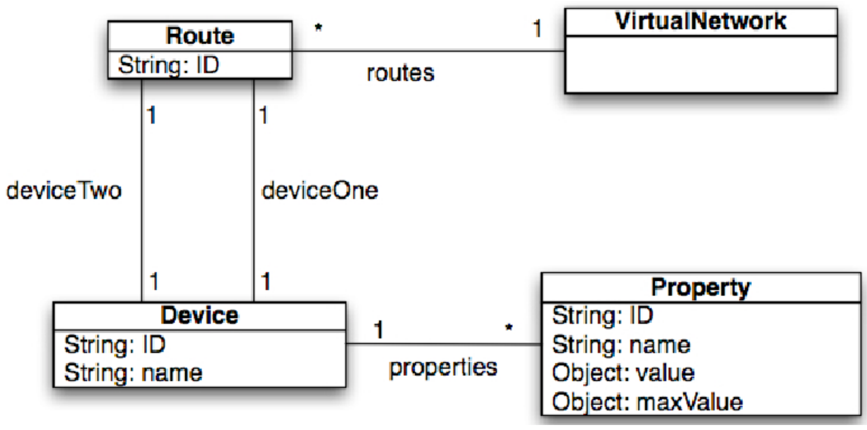


Figure 4.4: API Class Diagram

the setVirtualNetwork (addRoutes, removeRoutes) function receive as input two parameters: (i) the addRoutes parameter is a list of routes that will be part of the virtual network; and (ii) the removeRoutes parameter is a list of routes that will be removed from the virtual network. The following are the six functions that can be used to obtain information from the network:

- devices = getDevices() returns all devices that are part of the virtual network as descriptors used to refer to the devices in subsequent calls.
- device = getDevice(deviceID) returns the device descriptor identified by the deviceID. When the device does not exist it returns an invalid descriptor. The deviceID is only a symbolic link and must be managed by the virtual environment plugin.
- value = getPropertyValues(deviceID, propertyID) returns the value of the property identified by the propertyID from the device identified by the deviceID. When the property and/or device do not exist it returns an invalid descriptor.
- routes = getRoutes() returns the routing table of all devices as route descriptors. The routes are used to compute new virtual networks.
- routes = getRoute(deviceID) return the routing table of a given device identified by the deviceID.
- routes = getVirtualNetwork() return the set of routes that defines a virtual network.

More information about the use of the developed simulator and the corresponding demonstrator is given in Section 6.

Chapter 5

An Architecture for Adaptation of Virtual Networks on Clouds

Virtual networks (VN) are a new research topic advocated to increase flexibility, manageability and isolation in the Internet. However they introduce many open issues to become practical in real scenarios. On the other hand, cloud computing provides elasticity, where availability scales up on demand, with resources being offered frequently as virtualized services over the Internet. The use of virtual networks as a mechanism in cloud computing can provide traffic isolation, improving security and facilitating pricing. Also, it allows us to act in cases where the performance is not in accordance with the contract for services between the customer and the provider of the cloud. This chapter shows an architecture for the deployment of clouds over virtualized networks. This architecture, conceived within the scope of the Horizon architecture, was used to build a software infrastructure. This infrastructure was used to build a prototype on a testbed where experiments with virtual networks using the multi-agent system (Annex J - Report 4.2) were made. With this infrastructure we completed our initial testbed described in the previous task.

The proposed infrastructure allows the creation of virtual networks on demand, associated with the execution of workflows, isolating and protecting the execution environment. Also, it provides performance monitoring of virtual networks by acting preemptively in the case of performance dropping below the stated requirements. The management acts autonomously changing routes without interruption of services. To validate the proposed architecture, we built a prototype on a testbed, which we used to execute image processing workflows utilized in e-Science applications. We show results of real workflow executions in our testbed to evaluate the network performance, the overheads involved when using virtual routers, how virtual network links behave with data flow transmission, and how the adaptation provided by the virtual network management system can benefit the workflow execution.

5.1 State of art

Virtual networks are a new research topic advocated to increase flexibility, manageability and isolation in the Internet. However they introduce many open issues to become practical in real scenarios. These perspectives and research challenges are presented in [62]. In this chapter we explore the interfacing between infrastructure and service providers. Our scenarios use real services and a workflow application running in computational resources connected by virtual networks. The virtual network management system provides an interface to adjustments required by the workflow manager.

The implementation of virtual networks, its performance issues and trends, are addressed in [63]. We use its virtual machine approach to construct our virtual networks. Although our focus is not on performance, we could evaluate our results and assess if they are factual.

In [64], the authors present a resource management framework for VN-based infrastructure providers. In this work, an architecture called Local Resource Manager (LRM) was developed to monitor and control virtual resources in a physical machine and to provide an interface with external clients/agents to do a high-level management. They extend the Xen tools to enable a fine grain, self-adjusting virtual resources control. The evaluation was performed with an implementation of a mechanism for dynamic adjust of CPU resources based on the application requirements of QoS. In our work we are not interested in isolating and controlling the resources in the hosts of the computer environment, but the ones of the network that interconnect them.

Hao et al. [65] propose mechanisms to migrate virtual machines in clouds within different networks. As stated by the authors, this demands network reconfiguration to offer uninterrupted services for the cloud users, which is achieved through network virtualization. However, the authors do not evaluate performance issues when reconfiguring the virtual network.

This work contributes to the decision on how to reallocate data paths among different virtual networks in order to achieve an acceptable performance in a cloud computing infrastructure, as the one proposed in [65]. This reconfiguration is important in the virtual network management in order to efficiently use the available links by allocating virtual networks according to the current network usage needs. Such actions can help in obeying SLA contracts, giving priority to flows from users or applications with more strict requirements.

5.2 Workflow Management System

Our architecture is based on Service Oriented Computing (SOC) [66], and allows users to establish connections among services, organizing them as workflows. In order to enact real workflows in our experiments, the management of the service compositions in our infrastructure is made by the GPO (Grid Process Orchestration) [67], a middleware for service workflows execution in the grid (Figure 5.1). The GPO allows the creation and management of application flows, tasks, and services. The GPO uses workflows built with the GPOL (GPO Language) [67]. The GPOL is based on concepts of service orchestration from WS-BPEL [68], with the inclusion of specific directives for grids, such as state maintenance, potentially transient services, notification, data-oriented services, and groups. The language includes variables, lifecycle, fabric/instance control, flow control, and fault handling. Additionally, it allows the user to start task executions, service executions, and workflow executions in sequence or in parallel. The scheduling service is responsible for distributing the workflow services to be executed in the available resources. To accomplish this, the scheduling service may implement different algorithms with different optimization objectives, and decide which one to use depending on the application or current environment characteristics. Information about the available resources in the grid can be obtained through the resource monitor (RM). The RM operates in a distributed manner, maintaining one instance on each computing resource and providing on demand information for other services. Our middleware provides the monitoring of workflow executions. The GPO monitors the execution times of each section specified in the GPOL workflow, including the time spent on each operation invoked in the process, registering them in a log file exclusive for each workflow.

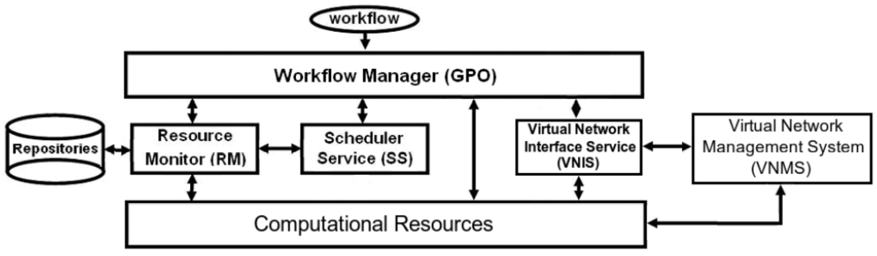


Figure 5.1: The Workflow Management Architecture.

The integration between the GPO and management of virtual networks is done through the Virtual Network Interface Service (VNIS). Using the VNIS, the workflow manager requests the network to be used for workflow execution. During the workflow execution the RM monitors the performance of virtual network links and can identify problems such as miscommunication or underperforming. In such cases, the GPO notifies the VNIS, requesting improvements in the performance of a link or the entire network when it is appropriate. VNIS offers functions to create a new VN, request an allocated bandwidth increase in the specific virtual link or an entire VN, provide information in real time from VN, and to release the VN associated with running a workflow.

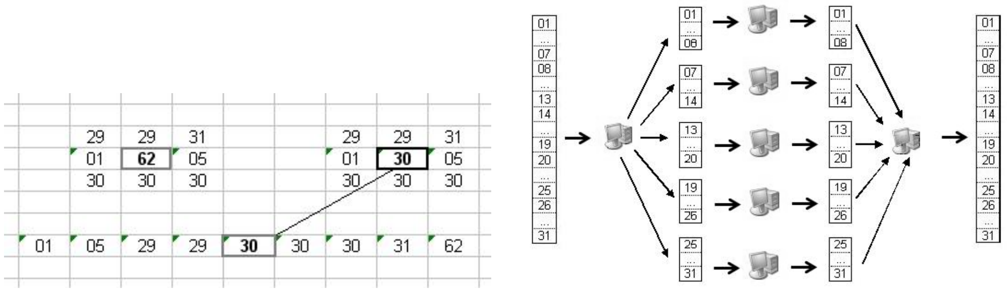
VNIS is the operational element that receives requests from the workflow management but that is not directly connected with the management of virtual networks. The interface between VNIS and the virtual networks is made by the Virtual Network Management System (VNMS) [69]. This modular design allows our system to use various alternatives for management of virtual networks. For each system we can use a specific version of VNMS without changing the other components. This flexibility allows our system to operate clouds in several domains simultaneously and transparently to the user. In this version of our testbed we are using VNMS to integrate our Workflow Management System with the Multi-Agent System presented in Annex J.

The system was implemented and deployed using the architecture proposed in Figure 5.1. Therefore, our experiments comprise the virtual networks, the workflow specification in GPOL, the workflow emulation using the emulator service over the GPO middleware, including workflow data transfers over different configurations of the virtualized network along with the adaptation provided by the VNMS component.

The main components of the proposed architecture (GPO, RM, SS, VNIS, VNMS) were modeled as services and implemented in Java [70]. The choice of transient services ensures scalability. For each workflow execution, an independent and exclusive VNIS instance is created. This instance is responsible for getting information and requesting actions from the virtual network created to support the workflow execution.

5.3 The Median Filter Application

The use of real applications in our testbed is essential, but there exist limitations to implement all the necessary services for all workflows and deploy them on all available resources. Such limitations include personnel and software requirements, which can be conflicting, making it not possible to experiment the necessary service-resource combinations to evaluate performance and strategies of network virtualization. To contour this situation, we created an emulation service which mimics many aspects of the workflows execution [71]. Using our emulator service we built emulation workflows which present a quite similar behavior to the real application workflows. In this work we used the median filter workflow to perform virtual network evaluations.



(a) Median filter (b) Workflow example

Figure 5.2: The median filter application.

The median filter is an image processing application [72] that can be executed in parallel by splitting the image into pieces and merging the results back into one single image. The median filter algorithm (Figure 5.2(a)) substitutes the value of a pixel by the surrounding values on its neighborhood. Figure 5.2(b) shows the file is divided into 5 parts, the slices files processed on parallel way, and the slices files merged on the final filtered file [67].

5.4 The Integrated Management of Resources

The management of our infrastructure works at two independent levels, which complement and interact when necessary. At the level of virtual networks, our multi-agent system based on the autonomic control loop and the knowledge base self-manages the virtual networks. If a failure occurs, it is diagnosed and repaired automatically. In addition, this level of management is also responsible for maintaining SLAs (Service Level Agreements) in its domain. If performance degradation is detected on any of the components of the virtual network, our multi-agents find alternative routes or use other routers to comply with the contract.

On the other hand, at the level of workflow management the execution of the services is followed step-by-step, measuring system performance from the user's perspective. For example, if the transfer time between computing resources is below the expected time, the manager can use VNIS to request more bandwidth in an attempt to keep the average performance in the execution of the workflow. This can occur even if the bandwidth is within the contract, because the overall performance of the workflow execution, in general, has higher priority over the contract at the network level. From the perspective of the user what really matters is the overall performance of the application as a whole.

However, to ensure the overall performance it is important to monitor at all levels of infrastructure. Our infrastructure performs such monitoring and can react quickly, independently and simultaneously at all levels of management.

Chapter 6

Demonstrators

The Horizon project produced many tools concerning network virtualization and autonomous network management and control. Some of the developed tools were already described in previous reports, as well as their demonstrations. Now, we provide a view of the demonstrators of the last developed tools and the integration among them.

Accordingly, when projecting a new virtual network, the administrator should design the autonomous control and test it using the developed agent simulator (Chapter 4). Based on these results, the administrator can develop his own agents using the Ginkgo platform associated to some of the developed tools. For instance, if the network virtualization platform is OpenFlow, then the administrator can program his agent using the agent platform provided in OMNI [35, 36]. Instead, if he uses a KVM platform, he could use the tools developed in this work package and presented in Report 4.2. Another more generic option would be to use the designed common interface for network virtualization (Chapter 3), so that the developed mechanisms are platform independent. The virtual network administrator could also think about network adaptive mechanisms, such as the ones described in Chapter 5. This mechanism allows the creation of virtual networks on demand and the execution of workflows, which provides isolation and protection in the execution environment. These mechanisms are easily integrated with the agent platform for a better performance.

Another important area is the development of the physical infrastructure, which provides the substrate for creating virtual networks. In Horizon Project, we developed management and control functions to develop testbeds using different virtualization platforms, such as Xen, OpenFlow, and KVM. We also developed a hybrid virtualization platform, which provides flexibility for the per-packet processing functions, besides supporting an easy flow management, which simplifies the migration of nodes and links.

In the next sections, we describe demonstrators of some of the developed tools and we show how they interact together.

6.1 Using SAVE in the developed testbeds

We now present some steps to understand how to create and manage a virtual network using the tools developed in Horizon.

6.1.1 Developing and managing a Xen testbed

A simple testbed consisting of a few nodes is enough to demonstrate all the features developed for Xen testbeds in Horizon Project, as show in Figure 6.1.

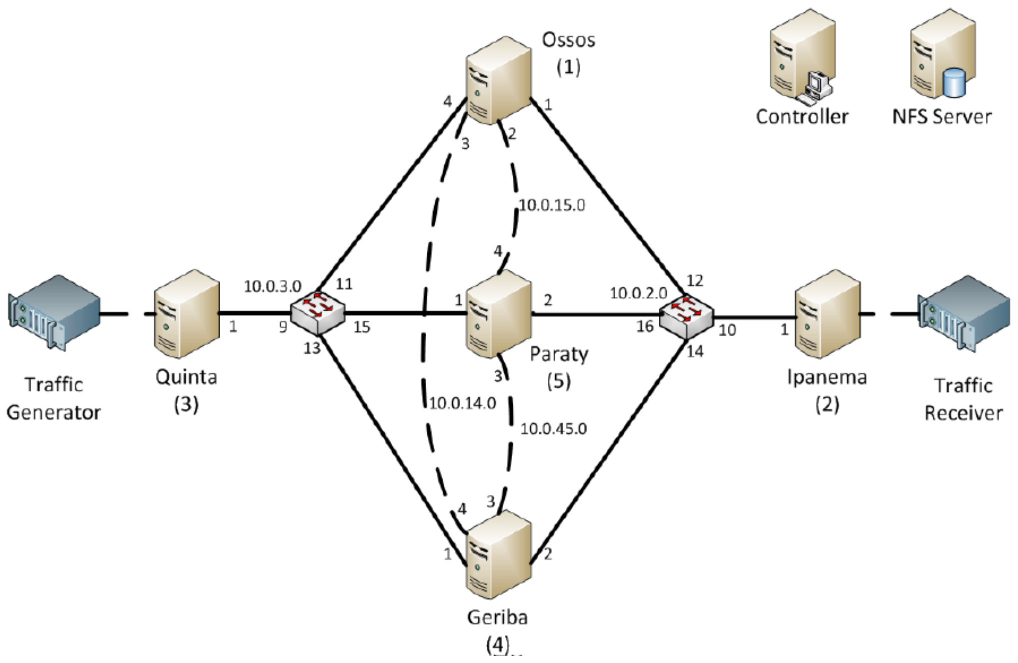


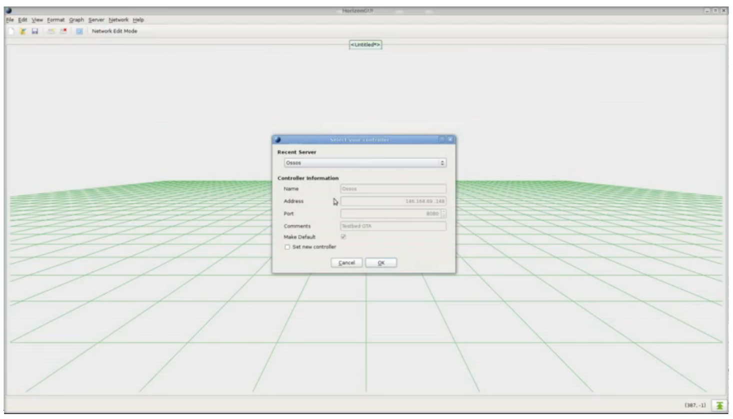
Figure 6.1: Testbed for creating and managing virtual networks using Xen and VNEXT.

The following steps allows the creation of a virtual network and the monitoring of a virtual node.

1. Build the testbed, having in mind that the controller, which represents VNEXT, and the NFS server must be connected to all physical machines.

2. Install VNEXT server in the controller and the client on the physical machines.

3. Open the management interface of VNEXT and connect to the testbed, as shown in Figure 6.2, by clicking in 'Server' and then in 'Connect'.



Recent Server
Ossos

Controller Information
Name: Ossos
Address: 146.164.69.148
Port: 8080
Comments: Testbed GTA
Make Default: ✓
☐ Set new controller

Figure 6.2: First, connect VNEXT graphical interface to the Xen testbed.

4. Verify whether all physical machines are connected and whether they are connect as described in Figure 6.1. Figure 6.3 shows the answer of VNEXT if all steps were correctly performed.

5. Click on 'Network edit mode' to deploy your virtual network. First, draw your virtual network, as described in Figure 6.4. After selecting the number and the position of the virtual nodes, draw the virtual links between them. Finally, click on the 'Deploy' button.

6. Repeat the previous steps to create more virtual networks.

7. Monitor any physical or virtual router by clicking on its representation on the visualization area of VNEXT. Figure 6.5 shows the data available for the selected physical router on the right-side menu. The left-side menu shows other options for managing physical and virtual nodes.

The developed live migration can be demonstrated by creating a flow between two external virtual machines. Hence:

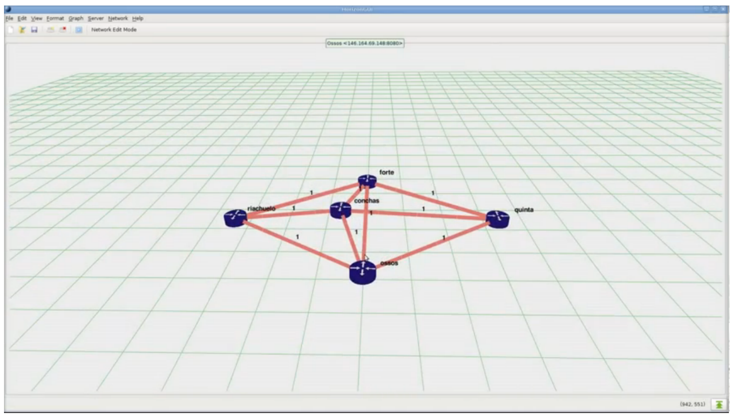


Figure 6.3: Physical topology view using VNEXT.

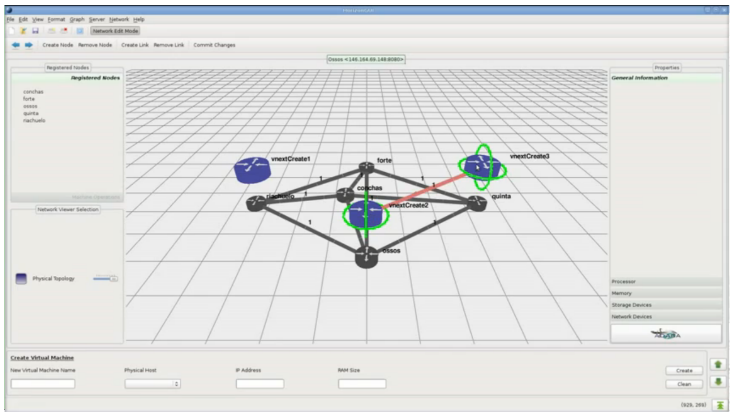
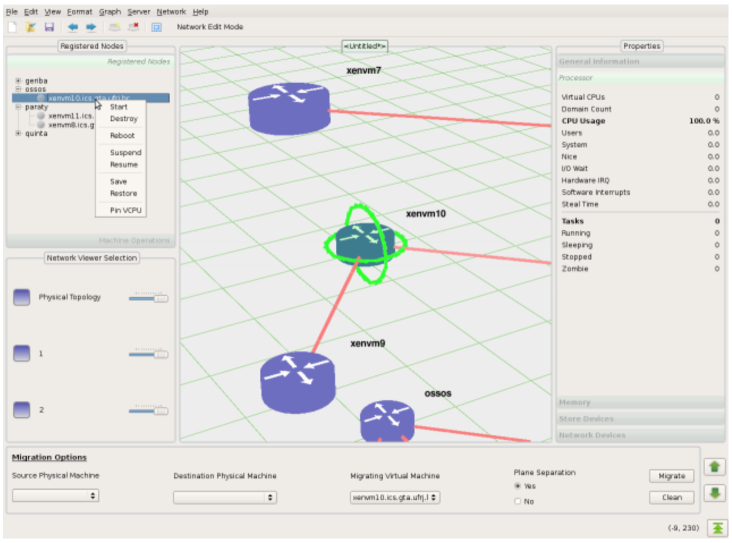


Figure 6.4: Create a virtual network by selecting the virtual topology and then requesting VNEXT to deploy the specified virtual network.

1. In the developed scenario (Figure 6.1), we can instantiate a video flow from the traffic generator to the traffic receiver, as shown in Figure 6.6.

2. Then, select the 'Migration' menu, in which it is possible to select the source physical machine, the destination physical machine, and the virtual machine to be migrated, as shown in Figure 6.6. After selecting the correct virtual machine, the administrator presses the 'Migration' button and VNEXT performs all action required to migrate the virtual machine and the virtual links without losing packets.

By using this procedure, the administrator observes that the video stream



Migration Options

Source Physical Machine

Destination Physical Machine

Migrating Virtual Machine: xenvm10.ics.gta.ufrj.b

Plane Separation: [✓ Yes] No

Virtual CPUs	0
Domain Count	0
CPU Usage	100.0 %
Users	0.0
System	0.0
Nice	0.0
I/O Wait	0.0
Hardware IRQ	0.0
Software Interrupts	0.0
Steal Time	0.0
Tasks	0
Running	0
Sleeping	0
Stopped	0
Zombie	0

Figure 6.5: Menu for managing virtual routers, which is available by clicking on the virtual node to be monitored.

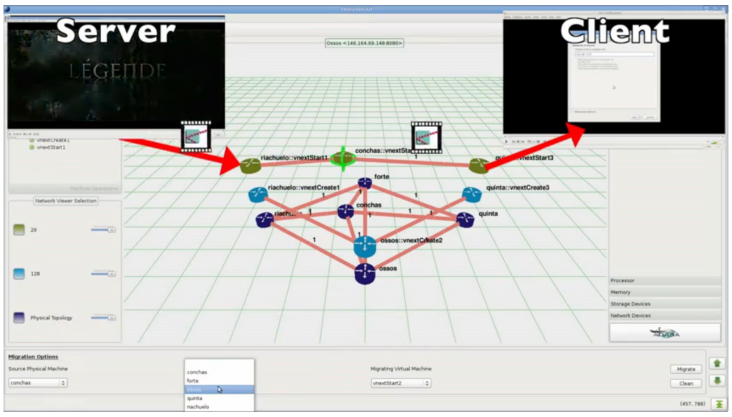


Figure 6.6: Instantiating a video flow between two external machines that uses a virtual network with three virtual nodes. The selected node will be migrated during the video flow.

is not affected by the virtual machine migration, which means that this operation is completely transparent to the end users.

6.1.2 Developing and managing an OpenFlow testbed

An OpenFlow network can be created using a set of interconnected physical devices or by creating a virtual network in a single machine, using machine virtualization techniques. Both methods can be used for performing this demonstration. We assume in this test a simple topology, as described in Figure 6.7.

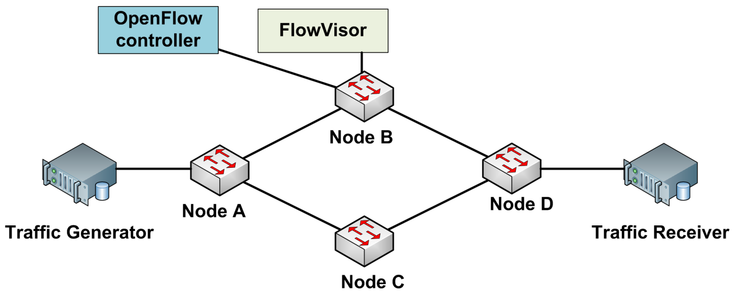


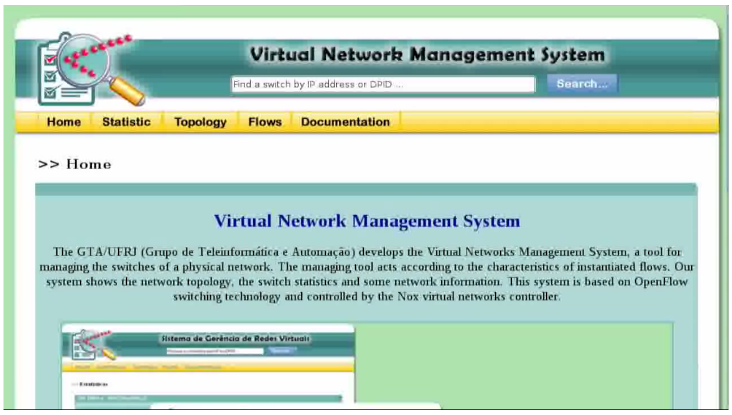
Figure 6.7: Testbed topology for the OpenFlow network running OMNI.

For performing a simple test of monitoring:

1. Create an OpenFlow network with the selected topology.
2. Configure FlowVisor, specifying the new virtual network.
3. Install and run the controller node with the OMNI version of Nox.
4. Open a browser and access the OMNI home page, shown in Figure 6.8.
5. The network visualization is done by clicking on the 'Topology' button. In the opened page, the administrator can visualize how the OpenFlow switches are connected and also how the spanning tree is configured in this network. Figure 6.9 shows a picture of this interface.
6. The network monitoring is performed by the service 'Statistics', available on the top menu. In this service, the administrator receives a list with all switches in the network, as shown in Figure 6.10.

   Each entry on this list is a link to the monitoring information about the switch, which includes the ports statistics about forwarded and lost packets and also the flow list with the corresponding monitoring data.

The migration in OpenFlow networks using OMNI is even simpler than the migration in Xen networks using VNEXT. Let's assume again that the


Virtual Network Management System
Find a switch by IP address or DPID ...
Search...
Home
Statistic
Topology
Flows
Documentation
Sistema de Gerência de Redes Virtuais

>> Home

Virtual Network Management System

The GTA/UFRJ (Grupo de Teleinformática e Automação) develops the Virtual Networks Management System, a tool for managing the switches of a physical network. The managing tool acts according to the characteristics of instantiated flows. Our system shows the network topology, the switch statistics and some network information. This system is based on OpenFlow switching technology and controlled by the Nox virtual networks controller.

Figure 6.8: OMNI home page, which provides access to all management functions.

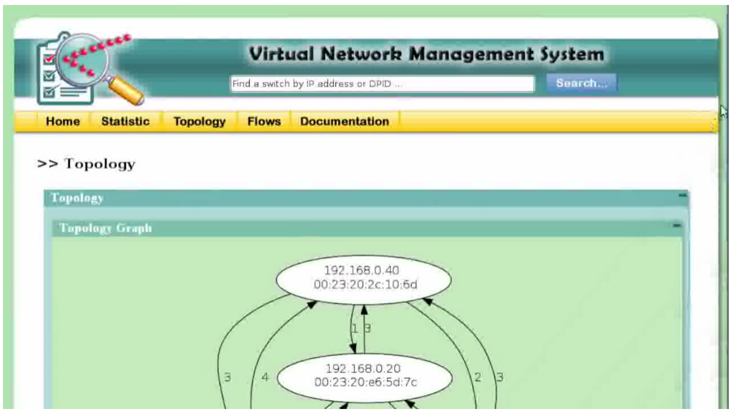


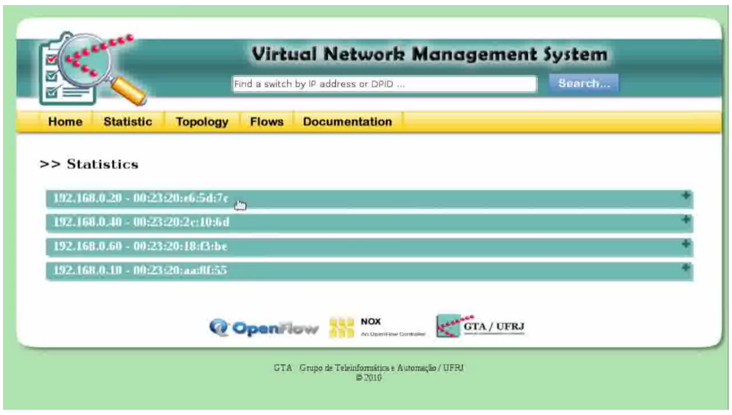
Figure 6.9: Network topology view using OMNI. This service is provided when the administrator selects the ‘Topology’ button in the Web interface.

traffic generator machine starts a video stream to the traffic receiver machine. To perform the migration and observe that the packets are not lost, the administrator do the following steps:

1. Start the video stream between the two machines.

2. Search for the video flow in the OpenFlow network. To do so, the administrator selects the ‘Flows’ button in the top menu, which opens the interface shown in Figure 6.11.

3. In this interface, type the UDP or TCP port used by the video in the ‘Port TCP/UDP Source’ field and start the search. OMNI will

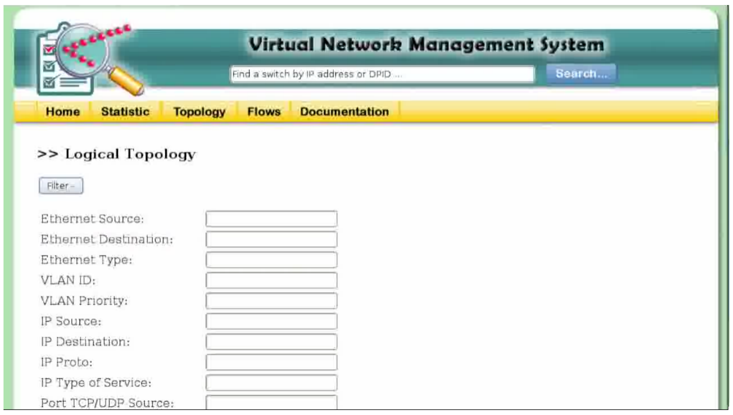

Virtual Network Management System
Find a switch by IP address or DPID ...
Search...
Home
Statistic
Topology
Flows
Documentation

>> Statistics

192.168.0.20 - 00:23:20:e6:5d:7c
192.168.0.40 - 00:23:20:2c:10:6d
192.168.0.60 - 00:23:20:18:f3:be
192.168.0.10 - 00:23:20:aa:ff:55

GTA Grupo de Teleinformática e Automação / UFRJ
© 2010

Figure 6.10: Monitoring the virtual network using OMNI. This service is provided when the administrator selects the 'Statistics' button in the Web interface.


Virtual Network Management System
Find a switch by IP address or DPID ...
Search...
Home
Statistic
Topology
Flows
Documentation
Filter

>> Logical Topology

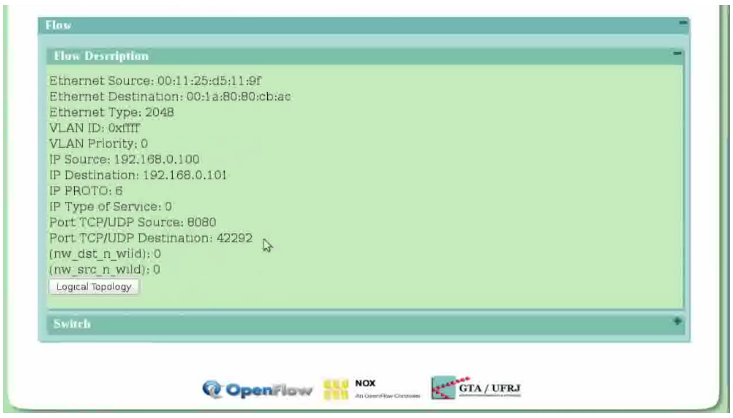
Ethernet Source:	
Ethernet Destination:	
Ethernet Type:	
VLAN ID:	
VLAN Priority:	
IP Source:	
IP Destination:	
IP Proto:	
IP Type of Service:	
Port TCP/UDP Source:	

Figure 6.11: Interface for searching for a specific flow in an OpenFlow network managed with the OMNI tool.

answer this request with a link to the video streaming flow. Click on the 'Logical Topology' button, in the bottom of the page, as described in Figure 6.12. This will open a new interface, containing the physical network and the path which is being used by the flow, as show in Figure 6.13.

4. Click on the button 'Select Path' and then select the new flow path in the network.

5. After selecting the new path, click on 'Create Path' to perform the



```
Ethernet Source: 00:11:25:d5:11:9f
Ethernet Destination: 00:1a:80:80:cb:ac
Ethernet Type: 2048
VLAN ID: 0xffff
VLAN Priority: 0
IP Source: 192.168.0.100
IP Destination: 192.168.0.101
IP PROTO: 6
IP Type of Service: 0
Port TCP/UDP Source: 8080
Port TCP/UDP Destination: 42292
(nw_dst_n_wild): 0
(nw_src_n_wild): 0
```

Figure 6.12: Flow description after a flow search. The Logical Topology button provides a graphical visualization of the flow.

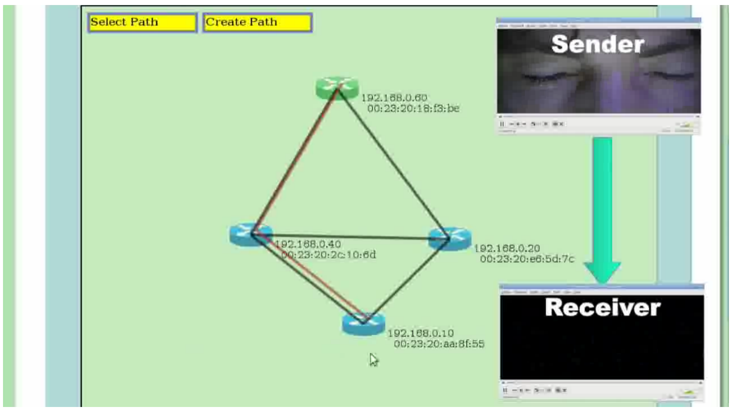


Figure 6.13: Visualization of the selected flow and interface to perform a migration.

migration without packet losses.

Again, the migration is transparent to the end user and no interruptions is observed in the video stream.

6.1.3 Using SAVE with Xen and OpenFlow

SAVE performs all the previous services with equal commands to both interfaces. This is the main advantage of SAVE, because the administrator does not need to reprogram the mechanism when changing the testbed.

Moreover, the administrator could use simultaneously two different testbeds using the same interface, because the underlying infrastructure is transparent to the virtual network operators in SAVE.

To perform the same virtual network management actions in both platforms using SAVE, a small set of functions are used:

1. Instantiate topology - Create the virtual network on the physical infrastructure, with the specified number of nodes and links, besides the specification of all dimensions. This function connects to the virtualization platform manager, in this case OMNI or VNEXT, and instantiates the virtual network by automatically performing all the required steps in each virtualization platform.

2. Get processor - Monitor the processor usage in the specified node. Other functions could be used to perform other monitoring actions in all the specified dimensions.

3. Modify topology - This function migrate a set of nodes and links as required by the infrastructure provider. Hence, this function calls the migration functions of OMNI and VNEXT and returns the received value.

Therefore, SAVE provides an easier and general way to manage virtual networks and virtualized physical testbeds.

6.2 Piloting plane simulator

In order to demonstrate the piloting system with the driving simulation environment, we implemented a virtual network in OpenFlow and their respective piloting agents in Ginkgo. For the sake of simplicity, we used a linear topology to demonstrate the application of the simulator and the proposed piloting plane. In Figure 6.14, we have the initial representation of the topology used. After the instantiation of the network elements, the agents begin the process of data collection. For this scenario, we have as a neighbor of router #1 the router #2 and vice versa. Therefore, the router #1 will run the Collect Behavior for collect its routing and cargo information, as well as the Request Information Behavior to request relevant data from the router #2, which responds through the Response Information Behavior. The router #2 performs the same collect process.

To begin the simulation process, the Client requests a stream service from the Stream Server, which is attended by a route composed by two

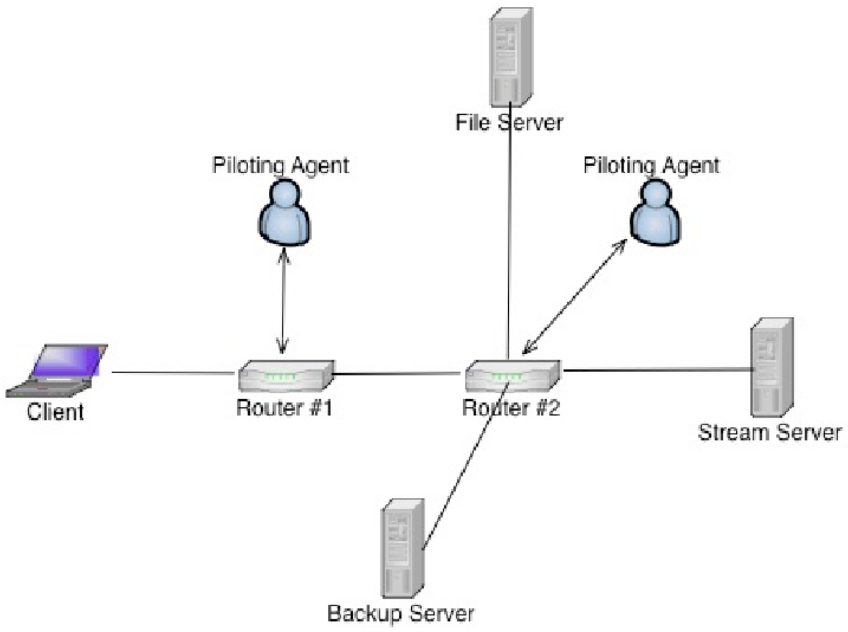
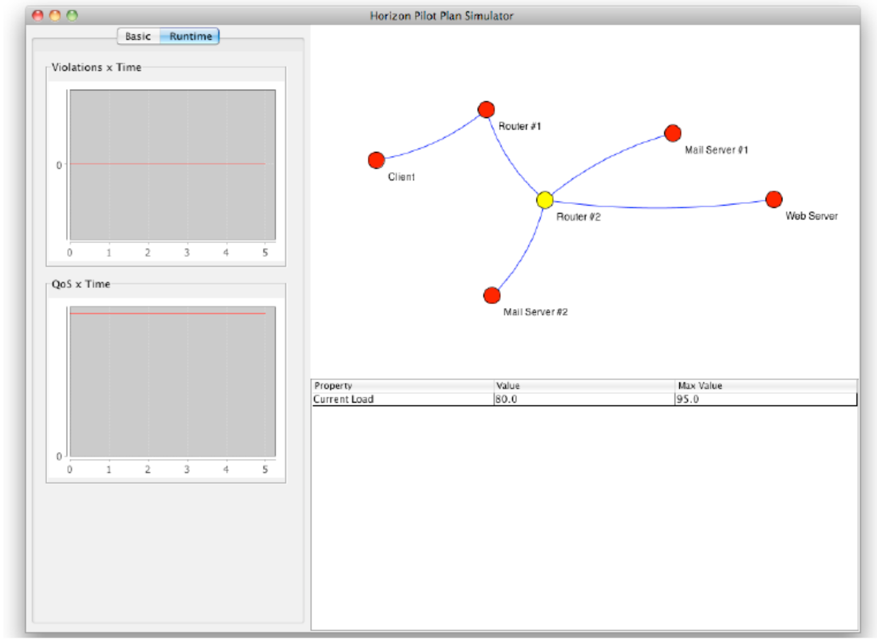


Figure 6.14: Virtual Network Topology.

virtual routers in the network. This stream request must meet certain criteria of quality of service (QoS), agreed in the Service Level Agreement (SLA). Initially, the routers are idle, and promptly attend to the requested stream with the required quality. We can observe in Figure 6.15 that the router #2 is in accordance with the QoS criteria.

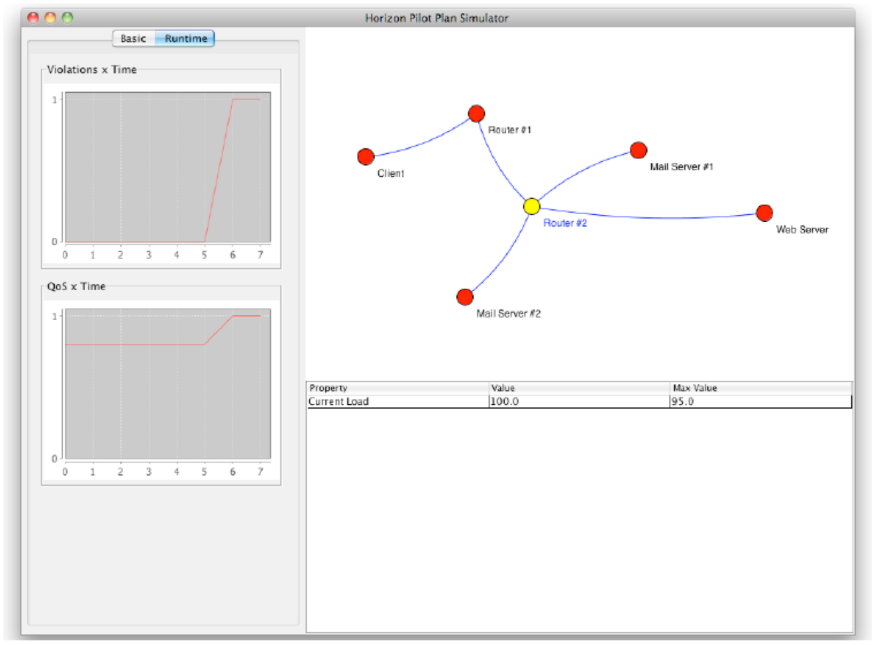
When starting transmission, the piloting agents verify the service quality attendance. However, to generate a disturbance in this scenario, we started a massive data transfer from the File Server to the Backup Server. Notably, this transfer will compromise the router #2, which becomes overloaded and unable to meet the quality required by the stream service, which initiates a process of quality norm violation. Figure 6.16 shows the router #2 overloaded and so violating the QoS norm.

Thus, the piloting agent realizes this disorder, and as a way to solve it starts searching for neighbors who have load available and meet the same segment of the route. However, as can be seen, no neighbor is able to meet this request. Thus, the piloting agent of the router #2, communicates with the Beacon controller, using the Create Router Behavior, to request the instantiation of a new virtual router on the network. After the construction of this new router #3, the piloting agent runs the Create Agent Behavior to allocate an agent to the router #3. Then the route in question, from router #1 to the Stream Server, is delegated to the router #3 through Route



Property	Value	Max Value
Current Load	80.0	95.0

Figure 6.15: Router #2 respecting the QoS criteria.



Property	Value	Max Value
Current Load	100.0	95.0

Figure 6.16: Router #2 violating the QoS norm.

Delegate Behavior, and finally the Inform Route Behavior informs the router

#1 about such modification in the route, as we can see in Figure 6.17.

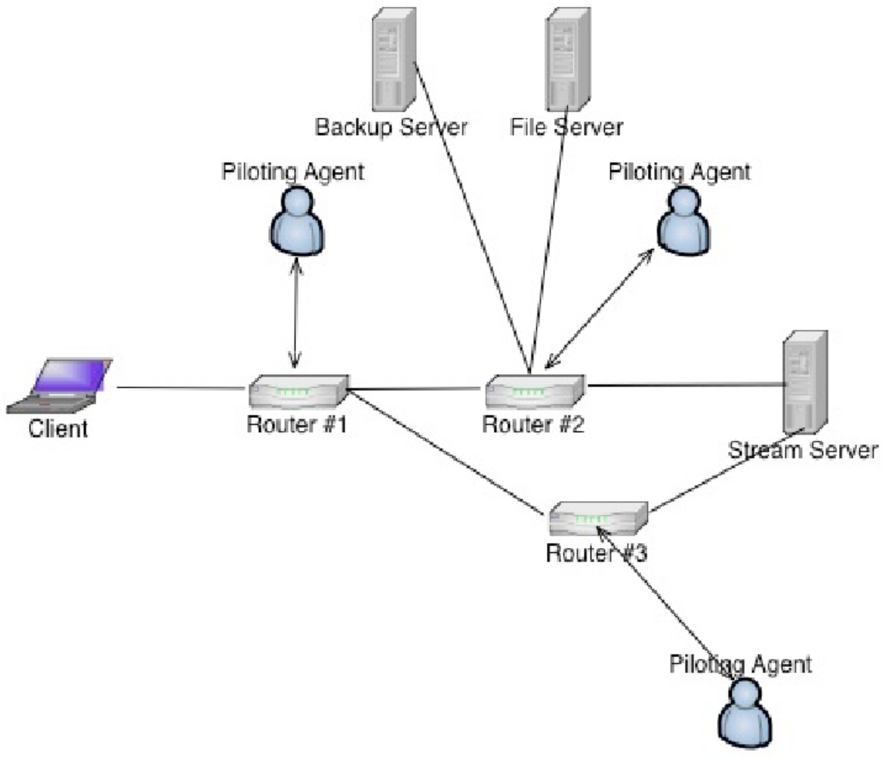


Figure 6.17: Final Topology.

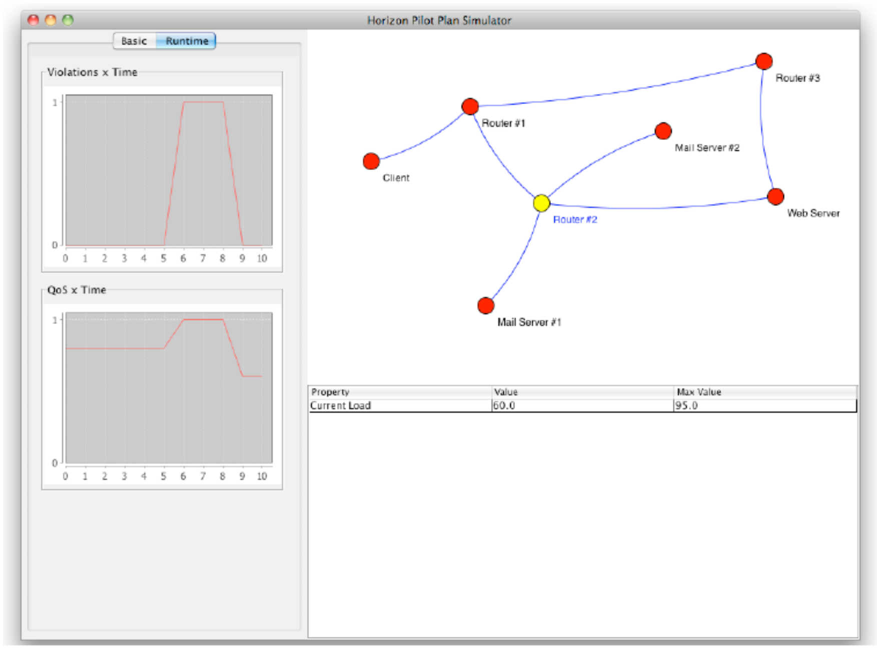
Through the Figure 6.18, we can view the information being displayed on the interface of the simulation environment. It is important to note that between the period of non-compliance of quality criteria and its solution, the piloting agent of router #2 is violating the norms of quality. Also, in the same figure we also note the fall of the QoS attendance of the transmission, and their improvement after the creation of router #3.

6.3 Adaptive environments

We performed real workflow executions in our testbed to evaluate the network performance in three aspects:

1. The overheads involved when using virtual routers.

2. How virtual network links behave with concurrent data flow transmissions.



Property	Value	Max Value
Current Load	60.0	95.0

Figure 6.18: Virtual Network fulfilling the QoS norm.

3. How the execution of workflows can take advantage of the adaptation provided by the virtual network management system.

The evaluations presented in this section are useful for the development of advanced management algorithms for the virtual network substrate. In addition, the experiments can provide background for the development of autonomic management agents capable of switching flows across network links when abnormal behavior is observed.

6.3.1 The Testbed Infrastructure

We deployed a testbed to execute our experiments using the virtual network. The infrastructure is composed of a network substrate, a set of software tools for creating on demand virtual networks, a computational grid, and a workflow management system. The testbed receives as input a set of workflows used to evaluate different strategies of network virtualization.

Our infrastructure uses the Globus Toolkit (GT) [73] deployment, an OGSA (Open Grid Service Architecture) [74] implementation. In the OGSA, all resources (physical or virtual), are modeled as services, bringing to the grid the concepts offered by Service Oriented Computing (SOC). Our base system is a GT version 4 deployed on 4 resources: *Apolo*, *Artemis*, *Hermes*, and *Nix*,

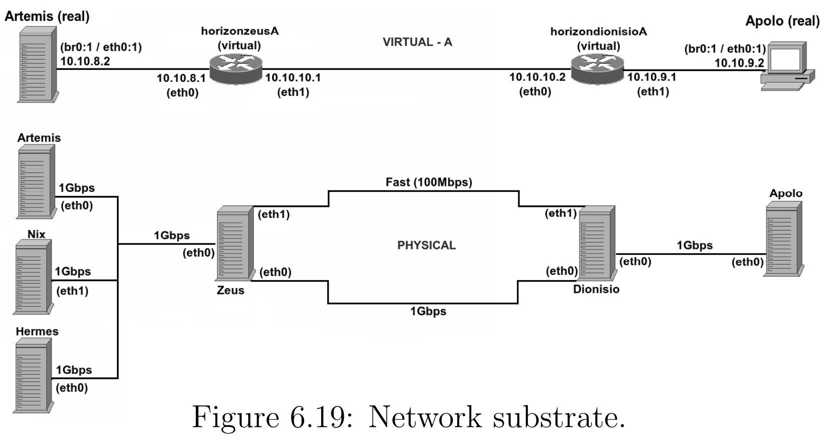


Figure 6.19: Network substrate.

Table 6.1: Resources in the testbed.

Name	**Processor**	**Clock**	**Cores**	**RAM**
Apolo	Pentium 4	3.00 GHz	2	2.5GB
Nix	Core2 Quad Q6700	2.66 GHz	4	8 GB
Hermes	Core2 Quad Q6700	2.66 GHz	4	8 GB
Artemis	Xeon 3040	1.86 GHz	2	1 GB

all with Debian Linux connected by the network substrate. This substrate is one of several possible customizations for our main network substrate. Resources characteristics are summarized in Table 6.1.

Each virtual network created in our testbed uses two virtual routers. These virtual routers are located at the real hosts *Zeus* and *Dionisio*, as shown in the Figure 6.19. For example, to bring virtual network A to operation, it is necessary to instantiate the virtual routers *horizonzeusA*, at the real host *Zeus*, and *horizondionisioA*, at the real host *Dionisio*. We instantiated 4 virtual networks to perform the experiments. Figure 6.19 also shows *Apolo* and *Artemis* connected by the virtual network A (IP 10.10*). Similar instantiations were made for the virtual networks B (IP 10.20*), C (IP 10.30*), and D (IP 10.40*). The paths for each virtual network can be mapped in one of two possible physical paths between the real hosts *Zeus* and *Dionisio*: a 100Mbps link and a 1Gbps link.

Merely summarizing what has already been detailed in Annex J/Report 4.2 for the developed multi-agent system, the main tools used to build our testbed are qemu, KVM, and libvirt. Qemu [75] is a processor emulator which can also be used as a virtualization platform. The Kernel-based Virtual Machine (KVM) [76] is a full virtualization hypervisor based on the machine emulator qemu. We use the KVM to create our virtual routers because it provides good performance in full virtualization, supports Symmetric Multi-Processor (SMP), memory ballooning, and live-migration of virtual machines. Furthermore, it allows VM networking by bridges, routing or private networks adding versatility to our testbed. KVM is free software under the GPL and open-source, and it allows the use of external tools to control it. The management of our virtual networks is made through Libvirt [77]. The Libvirt is an API to access the virtualization capabilities of Linux with support to a variety of hypervisors, including qemu, KVM, and Xen, and some virtualization products for other operating systems. It allows local and remote management of virtual machines. With Libvirt it is possible to use the same code to request information regarding the performance of a virtual link independent of the hypervisor running in the virtual routers.

6.3.2 Virtual Network Overheads

We start by evaluating the overhead introduced by the virtual routers when compared to the transmissions without them, i.e., in a switched gigabit Ethernet network. In this scenario we measured the times taken to execute a simple workflow which performs a median filter in an image. It uses 3 services and performs 2 data transfers, as shown in Figure 6.20.

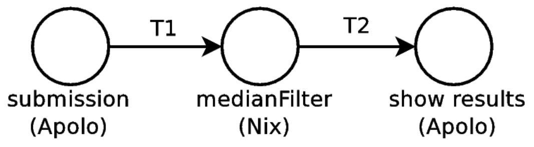


Figure 6.20: Workflow used in the virtual network overhead evaluation.

The workflow receives a user submission in *Apolo* and sends the image to be processed in *Nix* (transfer 1 - T1), where the median filter is applied to the image. After that, the image is copied back to *Apolo* (T2), where the resulting image is shown to the user. We executed this median filter workflow for images of $10,000 \times 10,000$ pixels. We compare executions of the workflow in a gigabit Ethernet network with the execution in our testbed using the gigabit links available from the virtual routers. Figure 6.21 shows the execution times of each workflow step averaged over 5 executions with confidence interval of 95%. In the next graphs, the label "LRC" refers to the data transmission using the substrate network directly, without any virtual element. (LRC is an acronym to *Laboratório de Redes de Computadores* – Computer Networks Laboratory).

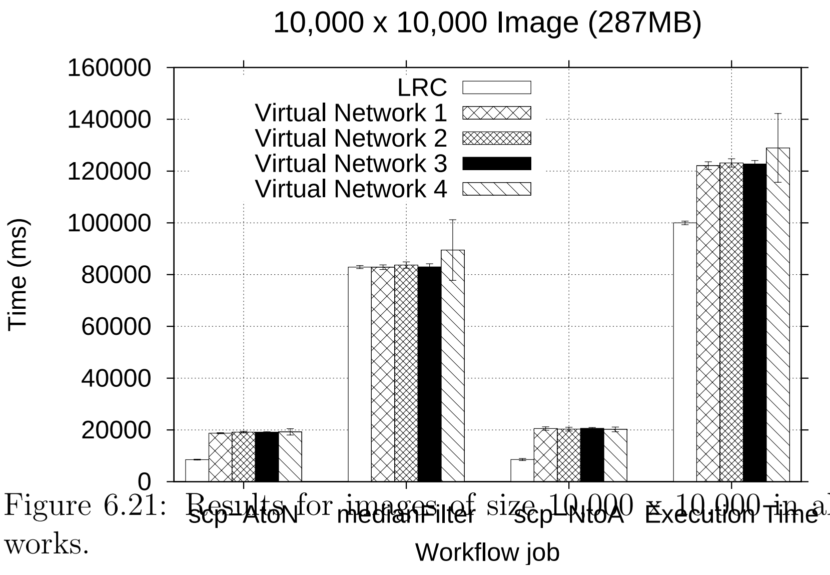


Figure 6.21: Results for images of size $10,000 \times 10,000$ in all available networks.

Figure 6.21 presents the execution times of the workflow services in all available networks in our testbed: 4 virtual networks plus the LRC gigabit network. We can observe that both data transfer services (scp-AtoN and scp-NtoA) double their times when using the virtual network to transfer the $10,000 \times 10,000$ pixels (287 MB) images. This impacts the final execution time of the workflow, which is increased by 23% (curves "Execution Time"). Therefore, the 4 virtual networks present similar behavior with some overhead over the LRC network.

The overhead introduced by the virtual routers when there were file transfers was expected [63]. This is caused by several factors, such as:

- Complexity in the packet forwarding through virtual machines;
- Multiplexing packets to virtual machines through bridges;
- While in the LRC network the data path between *Apolo* and *Nix* has a single hop, the virtual network transfer passes through 3 hops, introducing queue/propagation overheads to the data stream; and
- *Zeus* and *Dionisio* introduce overheads when processing the incoming data and forwarding it to the destination.

In this work we focus on the management of the flows through the available virtual networks, therefore we accept this overhead as part of the virtual network infrastructure.

6.3.3 Virtual Network Performance

In the previous section we evaluated the performance of a 1Gbps link in a virtual network with a single data flow. In this section we add a 100Mbps link, and we analyze the performance of both links using up to 3 data flows. Each data flow is a transfer of a $15,000 \times 15,000$ image file (644 MB). For such evaluation, we consider 4 routing scenarios (A, B, C, and D), as shown in Table 6.2.

Table 6.2: Scenarios used to evaluate the virtual network performance.

	100 Mbps			1 Gbps		
	Flow 1	Flow 2	Flow 3	Flow 1	Flow 2	Flow 3
A	**X**	**X**	**X**			
B		**X**	**X**	**X**		
C			**X**	**X**	**X**	
D				**X**	**X**	**X**

In each scenario, we measured the time taken to send all combinations of 3 flows, with each one being the data transfer of a $15,000 \times 15,000$ image file. The flows are as follows (Please, refer to Figure 6.19 to understand the topology of the network).

- **Flow 1**: TCP data transfer from *Nix* to *Apolo* using the virtual network 10.10.∗.
- **Flow 2**: TCP data transfer from *Hermes* to *Apolo* using the virtual network 10.20.∗.
- **Flow 3**: TCP data transfer from *Artemis* to *Apolo* using the virtual network 10.30.∗.

Figure 6.22 show results for scenario A (all flows routed through the 100Mbps link), where *Single* is the control measurement, i.e., the time taken for transferring each flow alone. We can note that the transmission of flows 1 and 2 concurrently (labeled "1+2") remains a little below the double of the control time, as expected. The same happens when only flows 1 and 3 and

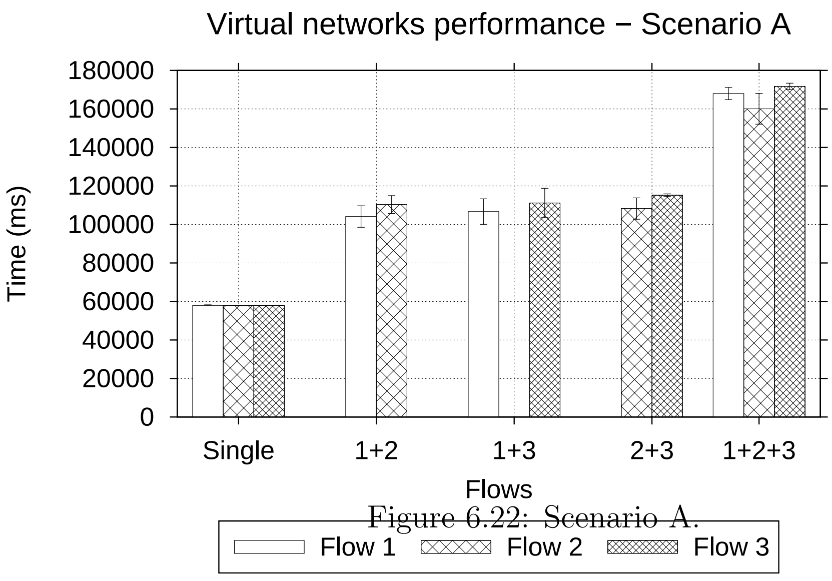


Figure 6.22: Scenario A.

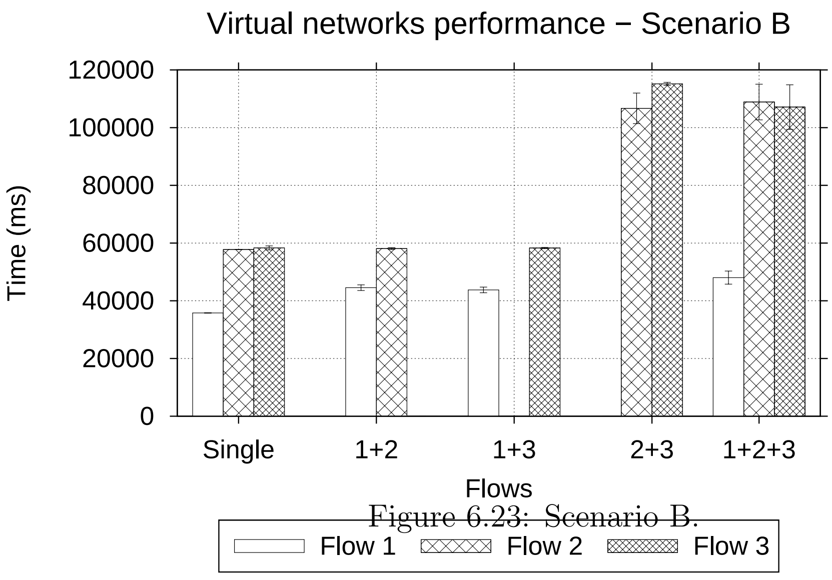


Figure 6.23: Scenario B.

when flows 2 and 3 are transmitted. When the 3 flows are transmitted, all of them have the performance significantly worsened by the concurrency.

When we consider scenario B (Figure 6.23), the control for flow 1 drops, as expected, since it is now in the gigabit link. Note that flow 1 has a similar transfer time with all combinations of flows, since it is always alone in the 1Gbps link. However, we can observe some overhead in flow 1 when increasing the number of flows due to processing concurrency in physical machines where virtual routers are instantiated. Flows 2 and 3 perform similarly to the control transfer when transmitted only with flow 1. When flows 2 and 3 are transmitted together, they share the 100Mbps link, what worsens their performance. By comparing scenarios A and B we note that changing flow 1 from the 100Mbps brings benefits to all flows.

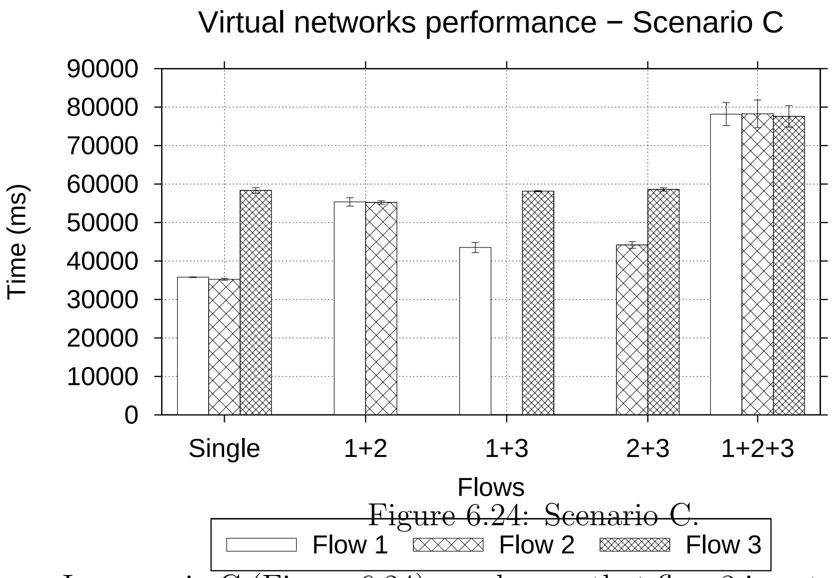


Figure 6.24: Scenario C.

In scenario C (Figure 6.24) we observe that flow 3 is not affected by flows 1 and 2, since flow 3 is the only one in the 100Mbps link, except in the case where all flows are sent together. This makes it clear that the concurrency in the physical machines where virtual routers are instantiated is also a limiting factor for the virtual network overall performance. In the "1 + 2 + 3" case, flows 1 and 2 affect each other when sharing the 1Gbps link, making their performance to be closer to the flow 3 alone in the 100 Mbps link.

When all flows are routed through the 1Gbps link (scenario D), the results in Figure 6.25 show that any combination of 2 flows results in a higher transfer time when compared to the control measurement, as expected. When all the 3 flows are transmitted, the concurrency makes the transmission time even higher. However, it is important to note that the transmission time of all flows together is smaller than the sum the transmission time of all flows alone. In addition, the transmission of the 3 flows concurrently in the 1

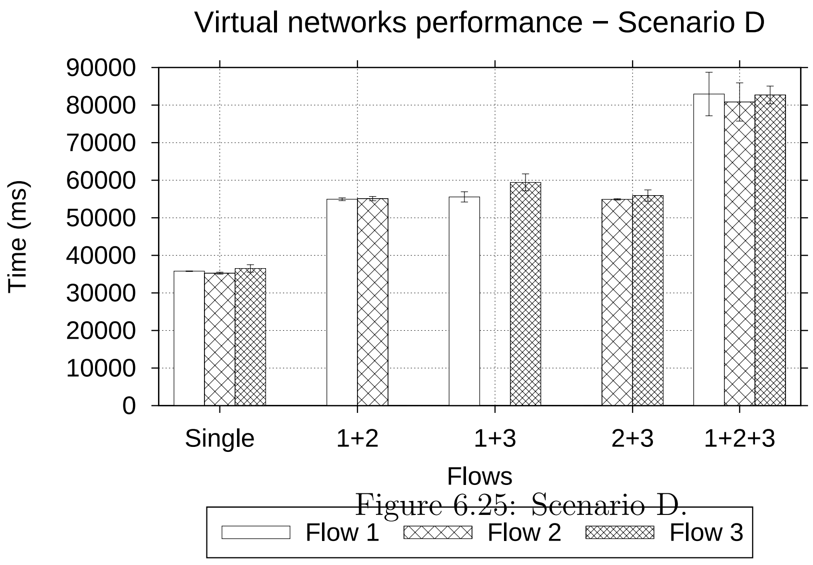


Figure 6.25: Scenario D.

Gbps network, i.e. scenario D, is faster than combinations in scenarios A and B. However, when compared to scenario C, scenario D presents similar performance when all 3 flows are being transmitted. Therefore, scenarios C and D are valid options for transmitting the 3 flows in the fastest manner in our testbed. These results can help in the development of adaptation strategies for executing workflows over virtual networks.

6.3.4 Network Adaptation

In this section we present results on how VNIS/VNMS could adapt the routing of flows during the execution to achieve a better performance. We use the same scenarios as in the previous section to refer to different distribution of virtual networks over the links.

We executed the median filter workflow splitting a $15,000 \times 15,000$ image file (644 MB) in 3 pieces, sending them to be processed in parallel on different resources, and receiving the 3 pieces back to generate the final resulting image (Figure 6.26).

The workflow steps are as follows.

1. *Apolo* breaks the image in 3 pieces.

2. *Apolo* transfers in parallel one piece to *Nix* using the virtual network 10.10.∗, one piece to *Hermes* using the virtual network 10.20.∗., and one

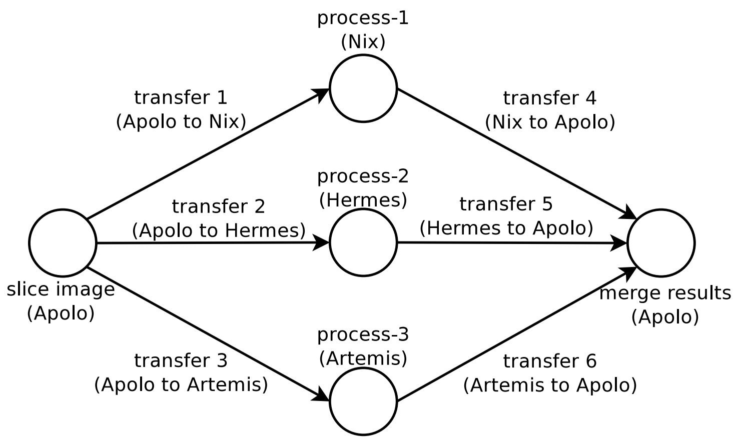


Figure 6.26: Workflow used in the virtual network adaptation evaluation.

piece to *Artemis* using the virtual network 10.30.∗. At this moment, all flows are routed through the 100Mbps link.

3. *Nix*, *Hermes*, and *Artemis* execute the median filter on their image pieces.

4. *Apolo* gets all the pieces back from *Nix*, *Hermes*, and *Artemis*. Here the flows are routed through different paths, using different scenarios as in the previous section.

Upon the execution of the workflow, the VNIS/VNMS can choose how to distribute the data flows in the virtual networks depending on the workflow's requirements. For example, if the transmission time for the first three transfers in the 100Mbps link is above the workflow requirements (given by SLA, for instance), the VNIS/VNMS can choose to migrate some networks to the 1Gbps link. Figure 6.27 shows potential gains of such adaptation when executing the workflow. We can observe that, if the VNIS/VNMS chooses to move from scenario A to scenario B, it would improve the transfer times for the returning image pieces (Transfers 4-6). As a consequence, the workflow execution time would also be reduced. However, if the workflow requirements are tighter, the VNIS/VNMS could choose to move to scenario C or D, achieving a data transfer time for the returning image pieces closer to the non-virtual network.

Figure 6.28 shows the time taken by each transfer and median filter processes. We can observe that transfers 4 to 6 show the same transfer time pattern achieved in the four scenarios. For example, *Transfer 4* is moved to the 1Gbps link in scenario B, having a transfer time close to the one presented by flow 1 in the previous section. This would be useful when there

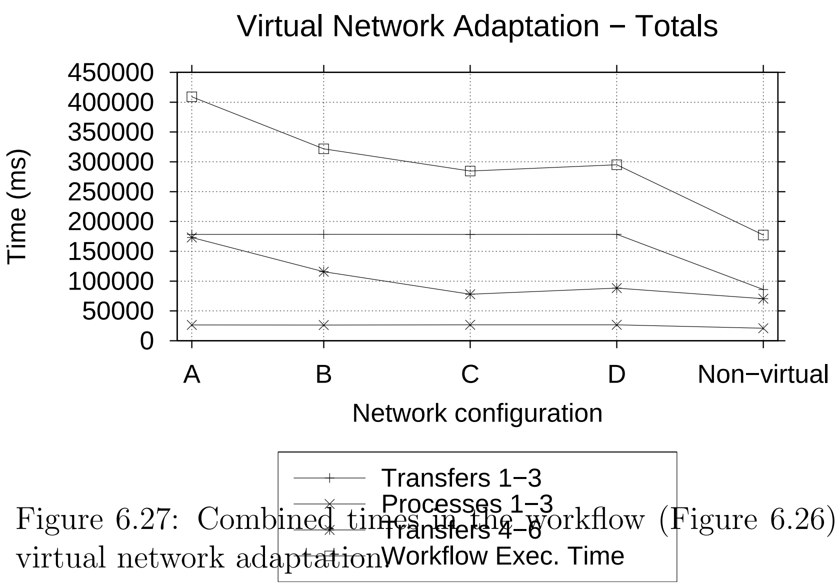


Figure 6.27: Combined times in the workflow (Figure 6.26) execution with virtual network adaptation.

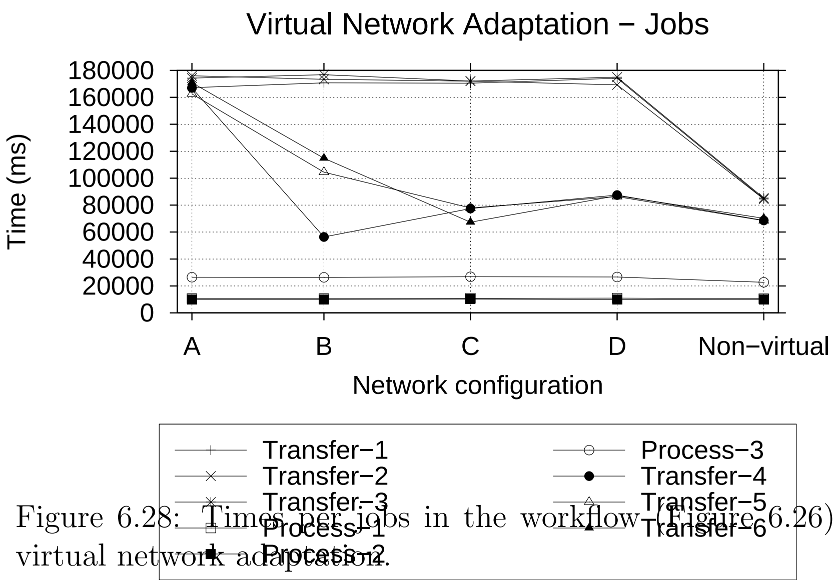


Figure 6.28: Times per jobs in the workflow (Figure 6.26) execution with virtual network adaptation.

is a priority flow in the network, which could be routed alone through the 1Gbps network. In scenarios C and D the transfer times for all transfers are similar, corroborating the experiments from the previous section by showing that both options are valid to more efficiently transfer three data flows.

6.3.5 Adaptation with Flow Priority

In this evaluation we present results of a case study on flow priority. We consider two workflows: (i) the workflow W_1 from Figure 6.29 for a $20,000 \times 20,000$ image (1.2 GB), which is split for processing; and (ii) a higher priority workflow W_P from Figure 6.20 with a $10,000 \times 10,000$ pixels image (287 MB).

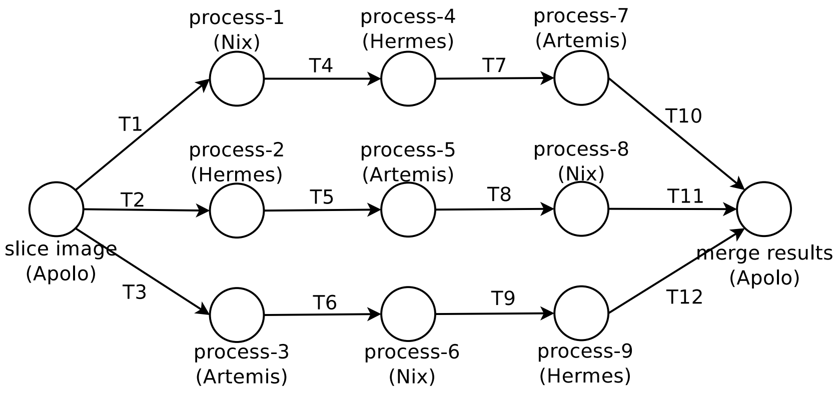


Figure 6.29: Workflow to apply three filters sequentially to the image file.

The case study was performed as follows. First, W_1 starts its execution, splitting the image file and sending its pieces to Nix, Hermes, and Artemis using the 1Gbps link provided by the virtual network (Please, refer to Figure 6.19 to understand the topology of the network). After the processing, W_P is submitted to execution in Apolo to be processed in Hermes. At this point, the network will experience concurrency among 4 data flows: $T2$, $T4$, $T5$ from W_1 and the first data dependency from W_P ($T1$) in the gigabit link. At this moment, the GPO requests priority to the VNMS through the VNIS. The objective of the VNMS now is to satisfy the higher priority from W_P. To achieve this, it must reconfigure the virtual networks to provide a faster transfer for *T1* from W_P.

We analyze 3 possible actions to be taken by the VNMS:

- **Action 1**: Take no action. Simply allow all flows to go through the 1Gbps link.
- **Action 2**: Reconfigure the network so that flow $T1$ from W_P can use the 100Mbps link exclusively.

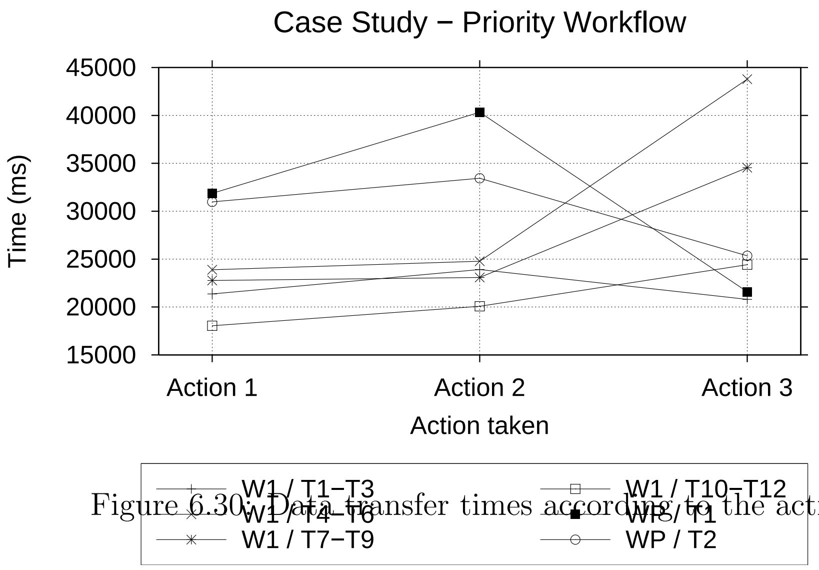


Figure 6.30: Data transfer times according to the action taken.

- **Action 3**: Reconfigure the network so that flow $T1$ from W_P can use the 1Gbps link exclusively, i.e., changing all the other flows to the 100Mbps link.

Results for data transfers in this concurrent workflow execution are shown in Figure 6.30. In the Action 1 case, the executions of the whole workflows (i.e., including all the processing times – not shown in the figure for the sake of cleanliness) take $206,038$ms for W_1 and $70,605$ms for W_P. Action 2 has shown to be the worse option for the priority workflow, since it worsens the data transfer time for its data dependencies (W_P/T1 and W_P/T2). In addition, in this case the total execution times for W_1 and W_P are $216,825$ms and $81,368$ms respectively. On the other hand, when Action 3 is taken, data transfer times from the priority workflow W_P are shorter, making the execution time of the whole workflow to drop to $50,598$ms.

This adaptation case study shows that, when a priority flow arrives, the best option is to route it through the gigabit link alone, as expected. However, as a second option, routing it through the gigabit link along with other 3 flows may still be better than routing the priority flow alone in the 100Mbps link.

Chapter 7

Conclusions and future works

The Horizon Project is focused on a new automatic piloting system which includes intelligent mechanisms to learn the context and adapt protocols to the current situation. The key idea of this research project is to introduce a new intelligent architecture adapted to virtual networks. The project is divided into four work packages: the context-aware post-IP architecture (WP1), the virtual network environment (WP2), the piloting plane (WP3), and the piloting of the virtual networking environment (WP4), which is the main focus of this report. The objective of the WP4 is to create the environment for the piloting system to supervise the work carried out in WP2 (Virtualization) and WP3 (The Piloting plane). Besides, the WP4 specifies the service control requirements establishing the basis for the WP activities in a form of policy based architectures or prototype applications. In this context the package was divided into three tasks: Internet service requirements analysis and case study (Task 4.1), overall system architecture design and testbed (Task 4.2), and test and optimization of the Horizon architecture (Task 4.3).

This deliverable presents the results of the Task 4.3. This Task, based on the requirement analysis produced by Task 4.1 and on the integration design specified in Task 4.2, has the objective of designing the overall virtualized and autonomous infrastructure with the piloting system. It is also responsible to realize an improvement on the virtual routers management and on the agent behaviors created in the Ginkgo platform. The contribution of this deliverable is the provision of a set of testbeds which can interact with each other and provide virtualization and autonomous management functions.

The management of virtual networks represents a challenge for the future Internet. Network operators should be capable of managing different substrate networks with different management interfaces. SAVE, presented in Chapter 3, proposes a solution to this challenge and allows network operators to simply operate virtual networks, despite of their inner particularities. SAVE categorizes network nodes in three possible configurations. `Forwarding elements` are nodes specialized in just forwarding packets, `processing elements` are nodes with specialization in processing operations, and `hybrid elements` combine packet forwarding with extra processing. The three node categories represent any possible configuration of network element. Nodes can be controlled by network primitives such as instantiating, monitoring, and migrating operations. Besides, their parameters can be defined in terms of processor, memory, bandwidth, topology, and traffic dimensions. SAVE adds a new layer between network operators and the virtual networks, thus network operators must know how to operate SAVE and SAVE automatically allows the management of the different virtualized substrates under the control of the operator. In order to prove the conformity of our mapping algorithms, we developed experiments to verify if virtual elements with different dimension requirements could be attended. Results show that the mapping algorithms ensure high conformity to Xen virtual nodes, guaranteeing the requirements of nodes. Also, we developed a prototype of SAVE in the Xen and the OpenFlow platforms, which is integrated with OMNI and VNEXT and allows a simple network management.

This last work package also improved the multi-agent system developed in Horizon. Chapter 4 presented a conceptual piloting system based on self-organizing and normative multi-agent system, which taking advantage of the intelligent decisions performed by piloting agents, needed to govern and adapt the virtual network in response to changing context. Besides the piloting system, we have implemented a simulation environment which supports users to test and analyze different normative and organizational configurations of the piloting multi-agent network.

In addition, in the Chapter 5 we presented our self-management system prototype, described in the report related to the task of work package 3.2, in which the concepts of autonomic networks were applied in a virtualized environment through a multi-agent system. In this autonomic self-management environment, experiments were performed with a focus on self-management failure, or self-healing, of virtual networks. Besides, in the Chapter 5, an architecture for the deployment of clouds over virtualized networks, conceived within the scope of the Horizon architecture, was presented. This infrastructure was used to build a prototype on a testbed where experiments with virtual routers and the Ginkgo platform were made.

Besides the work presented here is within the definitions of the Horizon Project, it is also aligned with the new paradigm of Networks as a Service (NaaS). The network virtualization can bring benefits to cloud computing as: aggregate traffic isolation, improving security, and facilitates pricing. This new mechanism permits, for example, to act in cases where the performance is not in accordance with the contract for services between the customer and the provider of the cloud.

We show how our infrastructure (network substrate, software, prototypes) can manage and adapt virtual networks on cloud environments. Our infrastructure allows the creation of virtual networks on demand, associated with the execution of workflows, isolating and protecting the user environment. The virtual networks used in workflow execution had its performance monitored by our manager which acts preemptively in the case of performance dropping below stated requirements.

To validate the proposed architecture, we built a prototype on a testbed to provide insights on how the virtual network management system can act to offer a better quality of service to the user. The results of image processing workflow executions showed that the management and adaptation of virtual networks are able to improve the data transfer times for the executed workflows.

Therefore, this report showed the last remarks about the algorithms, mechanisms, and tools developed within the context of the Horizon Project. We provided also demonstrations steps and results, which allows a visual observation of some of the achievements of this project.

7.1 Project Objectives

Project Horizon presented four main objectives, which were developed during the last 30 months. Now, we review these objectives, highlighting the solutions presented.

7.1.1 Objective 1

The first objective of Horizon Project was to outline the transition path from the today's real networks to virtual networks and define solutions for securing and controlling this network. This objective was achieved by the development of different testbeds with management and control support. Virtualization is the key for building a flexible network core that supports different innovative solutions. Our testbeds provides a conceptual proof that it is possible to build a network core completely based on network virtualization technologies. Indeed, we observed that we do not need the adoption of a single network virtualization platform, but of a single programming generic interface that can be used with any virtualization platform.

We also developed control algorithms that ensure security and quality of service for the virtual networks. We developed different algorithms that deal with several aspects of resource sharing in virtualization platforms. We even developed solutions that consider different management policies, in order to allow the infrastructure provider to choose the best solution depending on the service to be offered.

7.1.2 Objective 2

The second objective was to design and to test a piloting system associated with the virtualization paradigm to control the real resources distributed between the different virtual networks. This automatic piloting system controls the quality of service and manages the security of both global and individual networks. The objective is achieved by testing the algorithms defined in the first objective using the piloting system. Indeed, we design a piloting system that controls resources with a local and a global view. Hence, we control resource in the context of a node, assuring that each virtual node receives the agreed resources. We also guarantee that physical nodes will not be overcharged, by applying an access control of new virtual networks. In the global view, we developed mechanisms for detecting network failures and also for mapping new virtual networks in the physical substrate.

7.1.3 Objective 3

The third Horizon's objective was to introduce an intelligence-oriented network to define the different behaviors to automatically pilot the network. The idea was to introduce multi-agent systems with different behaviors to provide self-configuring, self-healing, self-optimizing, and self-protecting features in the network environment.

We developed multi-agent systems that work in different virtualization platforms. These multi-agent systems were implemented to provide the proof of concept when detecting and correcting network problems. Our agents perform adaptive services and are able to detect failures in the packet forwarding, solving the problems in the best way.

7.1.4 Objective 4

The fourth objective of Horizon was to optimize the quality of service and the security of the different virtual network using the piloting system. Indeed, the developed piloting system presents many algorithms and behaviors to deal with these issues. This optimization is based on precise monitoring functions, which provides the usage profile of virtual and physical nodes, besides providing specific measures required by the control algorithms.

The fourth objective is accomplished by the integration of the piloting system and the virtualized testbeds. By developing the testbeds and testing the control systems, we were able to optimize and integrate the developed solutions.

7.2 Future Work

Horizon Project results in many important contributions to science in the area of network virtualization and network autonomous control and management. The results respected and extended all the promised tasks. Nevertheless, the developed areas are still in progress and present many challenges that must be solved. Indeed, the work developed in Horizon also indicated some new steps that could be developed in new research initiatives. In the following, we present a summarized list of the topics that could be addressed in the future:

- Extend SAVE to other virtualization platforms. Other high level functions could also be added, including a graphical interface control;
- Integrate the developed testbeds by using the concepts of federation;
- Provide public access to the testbeds, by designing an access control entity;
- Design other security tools, different from network isolation, related to the provision of virtualized networks with autonomous control;
- Extend the set of autonomous functions for managing and controlling virtualized environments;
- Extend the agent simulation environment in order to enable the piloting agents to execute new behaviors and adopt new normative con-ducts. In addition, we intend to enable users to apply different self-organizing strategies and verify the system behavior in face of the application of such strategies; and
- Adapt all the developed agents to work with SAVE and, consequently, to interact with all developed testbeds.

7.3 Project impact

Horizon project introduced new concepts and tools, which include new virtual environments, network autonomic behavior, and piloting systems. All these results have a strong impact on innovation, dissemination, global standardization, and the implementation of future networks.

The Horizon project team developed an innovative autonomic-driven architecture. Moreover, this architecture allows that virtual networks activities in the area of protocol architecture are tested and used. Hence, our platform offers benefits in network virtualization that include increased flexibility, secure use and deployment, higher scalability, and more economically viable implementations.

Besides the technical achievements, the Horizon project also provided a competitive technological edge for France and Brazil in the Internet of the Future area thereby fulfilling the citizen's expectations about innovation in research areas of universities and enterprises. Hence, by its expertise and teamwork, this consortium contributed to a better collaboration between academia and industrial partners.

A final important remark about Horizon impact is that the project strengthened research in France and Brazil. Indeed, many professionals took part into Horizon and learnt about brand-new research areas. The project also increased the interaction between different teams inside each country and also between countries, forming new partnerships that will be repeated in other research experiences.

Bibliography

[1] “Horizon project: A new horizon to the internet,” 2011. http://www.gta.ufrj.br/horizon.

[2] S. C. Nelson, G. Bhanage, and D. Raychaudhuri, “GSTAR: generalized storage-aware routing for MobilityFirst in the future mobile Internet,” in *Proceedings of the sixth international workshop on MobiArch*, MobiArch ’11, (New York, NY, USA), pp. 19–24, ACM, 2011.

[3] A. Anand, F. Dogar, D. Han, B. Li, H. Lim, M. Machadoy, W. Wu, A. Akella, D. Andersen, J. Byersy, S. Seshan, and P. Steenkiste, “XIA: an architecture for an evolvable and trustworthy Internet,” Tech. Rep. CMU-CS-11-100, Department of Computer Science, Carnegie Mellon University, Feb. 2011.

[4] *Project SecFuNet - Security for Future Networks.* http://www.gta.ufrj.br/secfunet, Accessed in December, 2011.

[5] V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, N. H. Briggs, and R. L. Braynard, “Networking named content,” in *Proceedings of the 5th international conference on Emerging networking experiments and technologies*, CoNEXT ’09, (New York, NY, USA), pp. 1–12, ACM, 2009.

[6] D. K. Smetters and V. Jacobson, “Securing network content,” Tech. Rep. PARC TR-2009-1, PARC - Xerox, Oct. 2009.

[7] H. Project, “Technical reports,” 2011. http://www.gta.ufrj.br/horizon/index.php/technicalreports.

[8] N. Fernandes, M. Moreira, I. Moraes, L. Ferraz, R. Couto, H. Carvalho, M. Campista, L. Costa, and O. Duarte, “Virtual networks: Isolation, performance, and trends,” *Annals of Telecommunications*, pp. 1–17, 2010.

[9] N. Feamster, L. Gao, and J. Rexford, "How to lease the internet in your spare time," *ACM SIGCOMM Computer Communication Review*, vol. 37, no. 1, no. 1, pp. 61–64, 2007.

[10] N. C. Fernandes, M. D. D. Moreira, and O. C. M. B. Duarte, "Xnetmon: A network monitor for securing virtual networks," in *IEEE ICC'11 NGNI*, ICC'11 NGNI, 2011.

[11] G. Vallee, T. Naughton, and S. L. Scott, "System management software for virtual environments," in *Proceedings of the 4th ICCF*, (New York, USA), pp. 153–160, ACM, 2007.

[12] M. Bourguiba, K. Haddadou, and G. Pujolle, "Evaluating xen-based virtual routers performance," *International Journal of Communication Networks and Distributed Systems*, vol. 6, no. 3, no. 3, pp. 268–282, 2011.

[13] P. Padala, X. Zhu, Z. Wang, S. Singhal, and K. Shin, "Performance evaluation of virtualization technologies for server consolidation," *HP Labs Tec. Report*, 2007.

[14] R. S. et al., "Carving research slices out of your production networks with OpenFlow," *ACM SIGCOMM Computer Communication Review*, vol. 40, no. 1, no. 1, pp. 129–130, 2010.

[15] M. Armbrust, D. Patterson, A. Rabkin, *et al.*, "A view of cloud computing," *Communications of the ACM*, vol. 53, no. 4, no. 4, pp. 50–58, 2010.

[16] W. Ng, D. Jun, H. Chow, R. Boutaba, and A. Leon-Garcia, "Miblets: a practical approach to virtual network management," in *Proceedings of the Sixth IFIP/IEEE International Symposium*, pp. 201 –215, 1999.

[17] M. T. Jones, "Anatomy of the libvirt virtualization library - an api for easy linux virtualization," tech. rep., IBM, Jan. 2010.

[18] S. Wu, L. Deng, H. Jin, X. Shi, Y. Zhao, W. Gao, J. Zhang, and J. Peng, "Virtual machine management based on agent service," in *PDCAT'10 International Conference*, pp. 199 –204, Dec. 2010.

[19] D. Kim, J.-B. Kim, and J.-H. Park, "Virtual federated network operations on future internet," in *Proceedings of the 6th International Conference on Future Internet Technologies*, pp. 53–55, ACM, 2011.

[20] M. Pizzonia and M. Rimondini, "Netkit: easy emulation of complex networks on inexpensive hardware," in *Proceedings of the 4th ICST*, TridentCom '08, (ICST, Brussels, Belgium), pp. 1–10, ICST, 2008.

[21] K. M. Begnum, "Managing large networks of virtual machines," in *Proceedings of LISA '06: 20th Large Installation System Administration Conference*, pp. 205–214, Usenix, Dec. 2006.

[22] W. Fuertes, J. de Vergara, F. Meneses, and F. Galan, "A generic model for the management of virtual network environments," in *(NOMS), 2010 IEEE*, pp. 813 –816, april 2010.

[23] DTMF, "CIM system virtualization model white paper," Tech. Rep. DSP2013, Distributed Mangement Task Force (DTMF), Nov. 2007.

[24] P. S. Pisa, R. S. Couto, H. E. T. Carvalho, D. J. S. Neto, N. C. Fernandes, M. E. M. Campista, L. H. M. K. Costa, O. C. M. B. Duarte, and G. Pujolle, "VNEXT: Virtual network management for Xen-based testbeds," in *IFIP ICNF (to be published)*, pp. 1–5, Nov. 2011.

[25] D. M. F. Mattos, N. C. Fernandes, V. T. da Costa, L. P. Cardoso, M. E. M. Campista, L. H. M. K. Costa, and O. C. M. B. Duarte, "OMNI: OpenFlow management infrastructure," in *IFIP ICNF (to be published)*, pp. 1–5, Nov. 2011.

[26] M. Nascimento, C. Rothenberg, M. Salvador, and M. Magalhães, "QuagFlow: partnering Quagga with OpenFlow," in *Proceedings of the ACM SIGCOMM 2010 conference on SIGCOMM*, pp. 441–442, ACM, 2010.

[27] R. Sherwood, G. Gibb, K. Yap, G. Appenzeller, M. Casado, N. McKeown, and G. Parulkar, "Flowvisor: A network virtualization layer," tech. rep., Openflow-tr-2009-1, Stanford University, 2009.

[28] N. C. Fernandes and O. C. M. B. Duarte, "Xnetmon: Uma arquitetura com segurança para redes virtuais," in *Anais do X Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais*, SBSEG '10, (Fortaleza, CE, Brazil), pp. 339–352, Oct. 2010.

[29] N. C. Fernandes and O. C. M. B. Duarte, "Provendo isolamento e qualidade de serviço em redes virtuais," in *XXIX Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos - SBRC'2011*, pp. 1–14, may 2011.

[30] R. S. Couto, M. E. M. Campista, and L. H. M. K. Costa, "XTC: Um controlador de vazão para roteadores virtuais baseados em Xen," in *XXIX Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos - SBRC'2011*, pp. 1–14, may 2011.

[31] H. E. T. Carvalho, N. C. Fernandes, and O. C. M. B. Duarte, "Um controlador robusto de acordos de nível de serviço para redes virtuais baseado em lógica nebulosa," in *XXIX Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos - SBRC'2011*, pp. 1–14, may 2011.

[32] H. E. T. Carvalho, N. C. Fernandes, and O. C. M. B. Duarte, "SLAPv: A service level agreement enforcer for virtual networks," in *International Conference on Computing, Networking and Communications, Internet Services and Applications Symposium - International Conference on Computing, Networking and Communications, Internet Services and Applications Symposium*, pp. 1–5, jan 2012.

[33] P. S. Pisa, R. S. Couto, H. E. T. Carvalho, D. J. S. Neto, N. C. Fernandes, M. E. M. Campista, L. H. M. K. Costa, O. C. M. B. Duarte, and G. Pujolle, "VNEXT: Virtual network management for Xen-based testbeds," in *2nd IFIP International Conference Network of the Future - NoF'2011*, pp. 1–5, nov 2011.

[34] I. M. Moraes, P. S. Pisa, H. E. T. Carvalho, R. S. Alves, L. H. G. Ferraz, R. S. Couto, D. J. S. Neto, V. P. Costa, R. A. Lage, N. C. Fernandes, M. E. M. Campista, L. H. M. K. Costa, and O. C. M. B. Duarte, "VNEXT: Uma ferramenta de controle e gerenciamento para redes virtuais baseadas em Xen," in *Salão de Ferramentas do XXIX Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos - SBRC'2011*, pp. 1–8, may 2011.

[35] D. M. F. Mattos, N. C. Fernandes, V. T. da Costa, L. P. Cardoso, M. E. M. Campista, L. H. M. K. Costa, and O. C. M. B. Duarte, "OMNI: OpenFlow management infrastructure," in *2nd IFIP International Conference Network of the Future - NoF'2011*, pp. 1–5, nov 2011.

[36] D. M. F. Mattos, N. C. Fernandes, L. P. Cardoso, V. T. da Costa, L. H. Mauricio, F. P. B. M. Barreto, A. Y. Portela, I. M. Moraes, M. E. M. Campista, L. H. M. K. Costa, and O. C. M. B. Duarte, "OMNI: Uma ferramenta para gerenciamento autônomo de redes OpenFlow," in *Salão de Ferramentas do XXIX Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos - SBRC'2011*, pp. 1–8, may 2011.

[37] J. Turner and D. Taylor, "Diversifying the internet," in *Global Telecommunications Conference, 2005. GLOBECOM '05. IEEE*, vol. 2, pp. 755 –760, dec 2005.

[38] N. Feamster, L. Gao, and J. Rexford, "How to lease the internet in your spare time," *SIGCOMM Comput. Commun. Rev.*, vol. 37, pp. 61–64, January 2007.

[39] T. Anderson, L. Peterson, S. Shenker, and J. Turner, "Overcoming the internet impasse through virtualization," *Computer*, vol. 38, pp. 34–41, April 2005.

[40] J. Ferber, *Multi-Agent Systems: An Introduction to Distributed Artificial Intelligence.* Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1st ed., 1999.

[41] R. Brooks, "A robust layered control system for a mobile robot," *Robotics and Automation, IEEE Journal of*, vol. 2, no. 1, pp. 14 – 23, mar 1986.

[42] M. Berger and J. S. Rosenschein, "When to apply the fifth commandment: the effects of parenting on genetic and learning agents," *J. Exp. Theor. Artif. Intell.*, vol. 22, pp. 159–195, sep 2010.

[43] P. S. Dutta, N. R. Jennings, and L. Moreau, "Cooperative information sharing to improve distributed learning in multi-agent systems," *J. Artif. Int. Res.*, vol. 24, pp. 407–463, sep 2005.

[44] R. Tuomela, "The importance of Us: A philosophical study of basic social norms," tech. rep., Stanford University Press, 1995.

[45] R. Tuomela and M. Bonnevier-Toumela, "Social norms, task, and roles," Tech. Rep. HL-97948, University of Helsinki, 1992.

[46] R. Tuomela and M. Bonnevier-Toumela, "Norms and agreements," *European Journal of Law*, vol. 5, pp. 41–46, 1995.

[47] S. K. Mostfaoui, O. F. Rana, M. Ulieru, R. Valckenaers, and C. V. Aart, "Self-organisation: Paradigms and applications," *Engineering SelfOrganising Systems*, vol. 2977, pp. 1–19, 2004.

[48] K. Sycara, "Multiagent systems," *Artificial Intelligence*, vol. 10, no. 2, no. 2, pp. 79–93, 1998.

[49] H. Derbel, N. Agoulmine, and M. Salaün, “Anema: Autonomic network management architecture to support self-configuration and self-optimization in ip networks,” *Comput. Netw.*, vol. 53, pp. 418–430, February 2009.

[50] B. Jennings, S. van der Meer, S. Balasubramaniam, D. Botvich, M. O Foghlu, W. Donnelly, and J. Strassner, “Towards autonomic management of communications networks,” *Communications Magazine, IEEE*, vol. 45, no. 10, pp. 112 –121, october 2007.

[51] N. Sheridan-Smith, T. O’Neill, J. Leaney, and M. Hunter, “A policy-based service definition language for service management,” in *Network Operations and Management Symposium, 2006. NOMS 2006. 10th IEEE/IFIP*, pp. 282 –293, IEEE/IFIP, april 2006.

[52] J. Kephart and D. Chess, “The vision of autonomic computing,” *Computer*, vol. 36, no. 1, pp. 41 – 50, jan 2003.

[53] D. Chisnall, *The definitive guide to the xen hypervisor.* Prentice Hall Press, first ed., 2007.

[54] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, “Openflow: enabling innovation in campus networks,” *SIGCOMM Comput. Commun. Rev.*, vol. 38, pp. 69–74, March 2008.

[55] *JUNG - Java Universal Network/Graph Framework.* http://jung.sourceforge.net/, Accessed in July, 2011.

[56] G. Networks, “Ginkgo distributed network piloting system. white paper,” 2008.

[57] *OSGI Alliance.* http://www.osgi.org/Main/HomePage, Accessed in July, 2011.

[58] J. P. Perrone, *J2EE Developer’s Handbook.* Sam’s Publishing, first ed., 2003.

[59] *Jetty://.* http://jetty.codehaus.org/jetty/, Accessed in July, 2011.

[60] *JUnit.org Resources for Test Driven Development.* http://www.junit.org/, Accessed in July, 2011.

[61] *Apache Maven Project.* http://maven.apache.org/, Accessed in July, 2011.

[62] N. M. K. Chowdhury and R. Boutaba, "A survey of network virtualization," *Computer Networks*, vol. 54, no. 5, no. 5, pp. 862 – 876, 2010.

[63] N. Fernandes, M. Moreira, I. Moraes, L. Ferraz, R. Couto, H. Carvalho, M. Campista, L. Costa, and O. Duarte, "Virtual networks: isolation, performance, and trends," *Annals of Telecommunications*, vol. 66, pp. 339–355, 2011.

[64] F. Rodríguez-Haro, F. Freitag, and L. Navarro, "Enhancing virtual environments with qos aware resource management," *Annals of Telecommunications*, vol. 64, pp. 289–303, 2009.

[65] F. Hao, T. V. Lakshman, S. Mukherjee, and H. Song, "Enhancing dynamic cloud-based services using network virtualization," *Computer Communication Review*, vol. 40, no. 1, no. 1, pp. 67–74, 2010.

[66] F. Curbera, R. Khalaf, N. Mukhi, S. Tai, and S. Weerawarana, "The next step in web services," *Communications of the ACM*, vol. 46, no. 10, no. 10, pp. 29–34, 2003.

[67] C. R. Senna, L. F. Bittencourt, and E. R. M. Madeira, "Execution of service workflows in grid environments," *International Journal of Communication Networks and Distributed Systems (IJCNDS)*, vol. 5, no. 1/2, no. 1/2, pp. 88–108, 2010.

[68] OASIS, "Web services business process execution language version 2.0," tech. rep., OASIS Web Services Business Process Execution Language (WSBPEL) TC, April 2007.

[69] C. R. Senna, D. M. Batista, M. A. S. Jr., E. R. M. Madeira, and N. L. S. Fonseca, "Experiments with a self-management system for virtual networks," in *II Workshop de Pesquisa Experimental da Internet do Futuro (WPEIF 2011)*, (Campo Grande, MS, Brazil), Brazilian Computer Society, 2011.

[70] O. Corporation, "Java platform, standard edition (java se)," 2011. http://www.oracle.com/technetwork/java/index.html.

[71] C. R. Senna, L. F. Bittencourt, and E. R. M. Madeira, "An environment for evaluation and testing of service workflow schedulers in clouds (to appear)," in *International Conference on High Performance Computing & Simulation (HPCS)*, july 2011.

[72] C. A. Lindley, *Practical image processing in C: acquisition, manipulation and storage: hardware, software, images and text.* New York, NY, USA: John Wiley & Sons, Inc., 1991.

[73] G. Alliance, “Globus toolkit,” 2011. http://www.globus.org/toolkit/.

[74] I. Foster, C. Kesselman, J. Nick, and S. Tuecke, “The physiology of the grid: An open grid services architecture for distributed systems integration,” tech. rep., 2002. http://www.globus.org/research/papers/ogsa.pdf.

[75] F. Bellard, “Qemu, a fast and portable dynamic translator,” in *Proceedings of the annual conference on USENIX Annual Technical Conference*, ATEC ’05, (Berkeley, CA, USA), pp. 41–41, USENIX Association, 2005.

[76] I. Habib, “Virtualization with kvm,” *Linux Journal*, vol. 2008, February 2008.

[77] *Libvirt - The virtualization API.* http://libvirt.org/, Accessed in September, 2011.