Network Virtualization System Suite: Experimental Network Virtualization Platform

João Nogueira[1,2], Márcio Melo[1,2], Jorge Carapinha[2], Susana Sargento[1]

[1] Instituto de Telecomunicações, University of Aveiro, Portugal
[2] Portugal Telecom Inovação, Aveiro, Portugal
joaonogueira@ua.pt,marciomelo@av.it.pt,jorgec@ptinovacao.pt,susana@ua.pt

Abstract. Network virtualization arises as a potential solution for addressing the current IP limitations through a non-disruptive route, by letting multiple networks, running different protocols, to coexist and share the same infrastructure in an independent way. The purpose of this paper is to make this coexistence of networks a reality through the real support of multiple virtual networks. Therefore, this paper presents the Network Virtualization System Suite (NVSS): an experimental platform that integrates the functions of virtual network mapping and creation, discovery, monitoring, and management in a user-friendly platform. This platform contains novel mechanisms for network discovery and mapping: a distributed discovery algorithm of both physical and virtual nodes, and a heuristic algorithm for virtual resources mapping in the physical infrastructure, supporting the different characteristics of both links and nodes.
This virtualization platform is evaluated in different scenarios. The obtained results show the scalability and feasibility of the proposed mechanisms and functionalities (discovery, mapping and creation) in a single platform for network virtualization control and management.

Key words: Virtualization, Virtual Network, Network Virtualization, Discovery, Mapping, Embedding, Heterogeneous, Platform, 4WARD

1 Introduction

In the past decade, virtualization has increased in popularity and is currently a fundamental technological advantage on many IT businesses. Several companies spent a significant amount of resources into developing novel software and hardware solutions, supporting different levels of server virtualization, and have produced several successful and widely used commercial solutions.

Network virtualization, on the other hand, is still in an early stage. Although some virtualization technologies, such as Virtual Local Area Networks (VLANs) and Multi Protocol Label Switching (MPLS), have been developed and used on telecommunication networks, the full potential that can be achieved from a complete virtualized network environment is yet to be accomplished. A fully virtualized solution would be extremely flexible, by allowing multiple networks to run simultaneously, while supporting different technologies, protocols, topologies, and Quality of Service (QoS) requirements.

The advantages provided from a fully virtualized network can be significant [1], and therefore Future-Internet initiatives such as AKARI [2], GENI [3], and 4WARD [4], are assessing it as a viable and non-disruptive route to implement novel protocols and architectures.

Network virtualization has also drawn the attention of the Internet Service Providers (ISPs) [5] that look into virtualization as a technology that will decrease their infrastructure's Total Cost of Ownership (TCO). Additionally, network virtualization also presents business advantages, because custom-tailored client networks can be created and reconfigured quickly and on demand.

Although virtualization comes with many advantages, virtualizing a network presents many obstacles [6], not only technical but also business related. With respect to the technical difficulties, router virtualization is one of them: some solutions, such as the Virtual Router Project [7] and XORP [8], have been developed but, although the performance levels are promising, they are still not up-to-par with the current hardware offerings. Embedding virtual networks into substrate networks is another demanding and non-trivial undertaking that has been the target of several researches [9, 10, 11]. Furthermore, providing virtual network resource and topology discovery functionalities [12], and guaranteeing security and isolation between networks, have proven to be challenging tasks.

In the last few years, efforts have been made by the research community to build a network virtualization framework that could provide the community with a means of experimenting and validating solutions related to the previously referred issues. GENI is one such example: it provides researchers with a network framework that is highly programmable, and thus, can be used to test network virtualization concepts and algorithms. Despite being a mature and complex project, with multiple workgroups (including PlanetLab [13]), GENI by itself lacks some desirable functionalities, such as automatic creation of virtual networks, and topology discovery. It only provides the backbones upon which these mechanisms may be developed. The Virtual Network (VNet) Management Demonstrator [14] was built with the aim to provide a demonstrator capable of showing the creation of the VNets conceptualized in the 4WARD project. Although this demonstrator does indeed provide the functionality of virtual network creation, the network and creation processes are statically configured and based on specific configuration files and scripts. Despite its disadvantages, it also provides some potentially good features, such as the multi-threaded nature of the Agents. Therefore, this tool will be used as the basis for the developments and the platform presented in this paper.

In this paper, we present a complete experimental framework that allows the creation, monitoring, and management of virtual networks with arbitrary topologies, as well as virtual resources with any number of virtual interfaces. Moreover, it contains functionalities, such as: (1) centralized and distributed Virtual Network (VNet) discovery algorithms, (2) a heuristic mapping algorithm of embedded virtual networks, that optimizes both links and nodes mapping, (3) automated VNet creation, (4) dynamic resource monitoring, and a (5) Graphical User Interface (GUI) that integrates all these features and significantly improves

the user experience. We also present the supported scenarios of the virtualization platform and the results achieved with the experimentation of the proposed mechanisms in the real testbed. The presented results show the scalability and performance properties of the developed platform and respective mechanisms.

This paper will start with the discussion of the architecture of the NVSS on Section 2, followed by the implementation details on Section 3. Afterwards, an overview of the supported scenarios will be provided, on Section 4. Section 5 will be devoted to experimental testing, after which the paper will conclude on Section 6.

2 Network Virtualization Architecture

2.1 Business Model

In a fully virtualized network, an abstraction layer, besides enabling the concurrent existence of multiple networks on a single substrate network, also enables the decoupling of the services from the infrastructure. Due to this extra degree of freedom, i.e. being able to run services in an infrastructure-independent way, the decoupling of the traditional ISP role into two or more roles seems logical.

In order to take advantage of this decoupling, the 4WARD project defined a business model composed of 4 main roles: the Infrastructure Provider (InP), who is in charge of providing and managing the infrastructure; the Virtual Network Provider (VNP), that is responsible for assembling virtual resources, from one or more InPs, in order to build the virtual topology; the Virtual Network Operator (VNO), who installs and operates the VNets provided by the VNP; and, finally, the Service Provider (SP), who is solely responsible for using the VNet and providing services to its users.

The business model's roles and relationships are illustrated on Figure 1.

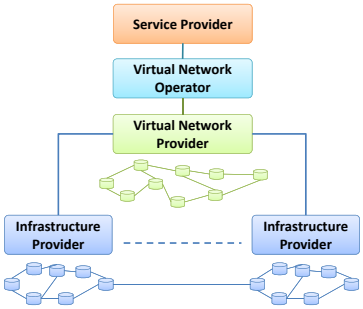


Fig. 1. SP, VNP, VNO and InP business model.

2.2 Architecture

The NVSS is composed of 3 software modules: the *Agent* module, the *Manager* module and the *Control Centre* module; their hierarchical decomposition is demonstrated in Figure 2. The Agent module is designed to run on every substrate node, in order to act upon it and periodically gather data from it. The Agents, besides interacting with each other, receive and send requests to

the Manager, which is a centralized entity in charge of aggregating all Agents' knowledge and sending them commands. Additionally, the Manager also communicates with the Control Centre, which is the user's front-end, and provides him with graphical and simple to use virtual network creation, management, and monitoring functionalities.

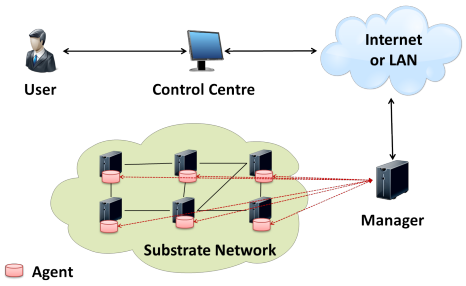


Fig. 2. Global architecture overview.

2.3 Functionalities

Distributed Physical and Virtual Network Discovery Physical and virtual network discovery provides a fast and easy way of having a global view of the virtual networks running on top of a given physical infrastructure, and it is also fundamental for the process of embedding new virtual networks, since the embedding process requires an accurate and up-to-date view of the substrate and currently running virtual networks.

The experimental virtualization platform works both with centralized and distributed discovery mechanisms. In the centralized version, the Manager requests information from the agents to build the topology. In the distributed version, the Agents perform the discovery by themselves without the Manager's interaction. The distributed discovery mechanism is able to reduce the required computing power on the central management node.

The proposed algorithm is based on concepts from both Spanning Tree Protocol (STP) and Border Gateway Protocol (BGP). At startup, the Agents register themselves in a pre-configured link-local multicast group, and begin exchanging messages with each other afterwards. The distributed algorithm for full network discovery relies on the neighbourhood concept, where each physical node knows exactly who its neighbours are, and also who are the virtual neighbours of its local virtual nodes. By aggregating each Agent's knowledge, the full topology map may be built. The concept is demonstrated in Figure 3.

Additional information about the discovery algorithm, experimental and simulation results, may be found in [12].

Virtual Network Creation The Control Centre module gives the possibility for the user to draw a new virtual network, by selecting and placing resources on the platform GUI and by connecting them with links. The user may specify the resources CPU capabilities, RAM amount, location, number of interfaces and also perform network addressing configurations. The final step in creating a new

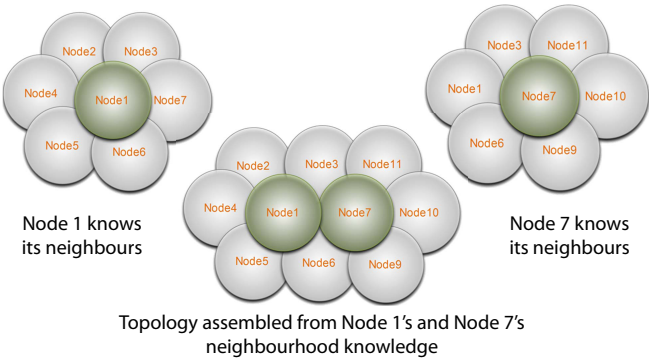


Fig. 3. Distributed Topology Discovery: Assembling neighbourhood knowledge.

virtual network is to commit it to the Manager, which will then map it in the physical infrastructure.

The embedding problem is a complex one (NP-hard) and a trade-off had to be chosen between computation time and embedding optimization. In order to lower the computational requirements that a dual-optimization algorithm requires (nodes and links), a heuristic mapping algorithm was developed, which aims to embed virtual networks taking into consideration both the substrate links and nodes loads.

The mapping algorithm, which is thoroughly discussed in [9], starts by determining a set of possible physical candidates for each virtual node to embed, based on CPU, RAM, hard drive and location constraints. After attaining a set for each virtual node, node stresses are calculated for each physical candidate, based on the concepts of [10]. These node stresses provide an indicator of the resources load: they take into consideration not only the resources capabilities, but also their active virtual machines and free resources, such as available RAM amount and available CPU load. A stress factor is also calculated for the physical links that takes into account both the total link bandwidth and the link load. By combining both stress factors, for each virtual node, a physical candidate node is chosen so that it offers the best compromise between node stress and link stress for its potential physical neighbours. As soon as every virtual node gets a unique physical node mapping, a Constrained Shortest Path First (CSPF) algorithm is run between mapped physical nodes in order to assess the best constrained paths to establish the virtual links. If the mapping succeeds, i.e. if every virtual node has been assigned to a unique physical node and every virtual link's physical path has been successfully determined, the Manager will then inform the Control Centre, and proceed with the effective creation of the requested virtual network, by sending link and node creation commands to the relevant Agents. Otherwise, the user will be informed that the mapping has failed and will include the reason for this failure.

Substrate and Virtual Network Monitoring A dynamic resource monitoring feature is required to have an accurate view of the virtual and physical networks, and quickly react to failures or configuration problems. The implemented

monitoring functions periodically update the resources' information; therefore, it is possible to identify diverse situations, such as failures and high resource usage, which may require immediate action.

Every Agent periodically checks its local resources' configuration and status, and reports back to the Manager if any change occurs. Several parameters are monitored: CPU load, RAM, HDD usage, interface and link status, interface bridge attachment and configuration, number of running virtual machines and their state.

Virtual Network Management The management feature is also very important; to that end, some functionalities are provided, like the change of the resource's state, i.e.: reboot, shutdown, suspend or power up, the change of the assigned RAM memory in runtime; and the deletion of either a single resource or a complete virtual network, which greatly simplifies the administrator work.

3 Experimental Platform Implementation

3.1 Control Centre Module

As previously stated, this is the module in charge of the user interaction, the GUI. This module was written in Java and is composed by two main threads: the Model and the View thread (Figure 4).

The Model thread is mainly used for listening to the Manager's messages and, aside from the exception of requesting the Manager an ID at start-up, it is a passive thread. Its main function is to process the Managers' messages and update the related databases. The communication process is based on blocking TCP/IP sockets.

As for the View thread, it is responsible for the actual graphical environment, and resorts to Java's *SWT* graphical API. It is in charge of handling user interactions, of displaying the networks' topologies and resource information, and of sending requests to the Manager. All of the platform's functionalities are presented to the user through the View thread.

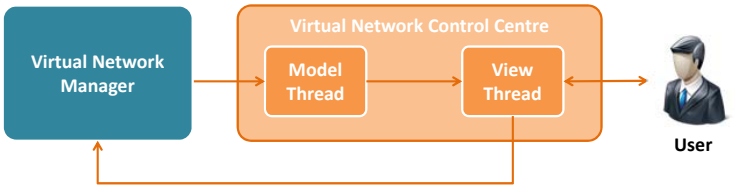


Fig. 4. Control Centre Module Decomposition.

3.2 Manager Module

The Manager is a central module, written in C, whose job is to assemble the entire network's information, provide unique IDs to each Agent and Control Centre, and to act on the networks. The connection between Agents and Control Centres is established through TCP/IP sockets.

In order to communicate with the Agents, a single *Command Send Thread* exists for sending messages, while multiple *Agent RX Threads* exist for receiving Agent messages. As for the connected Control Centres, a *Control Centre RX Thread* and a *Control Centre TX Thread* is created for each one, so that bidirectional communication may take place.

Besides the connection handling threads, there is an additional thread, the *Status Update Thread*, used to process the received resources' and pending links' information, update the Manager's databases and trigger Control Centre updates: if a resource or link update is detected in a virtual network previously requested by one or more Control Centres, an update will be sent via the respective Control Centre's TX thread.

Figure 5 summarizes the existing threads on the Manager's module.

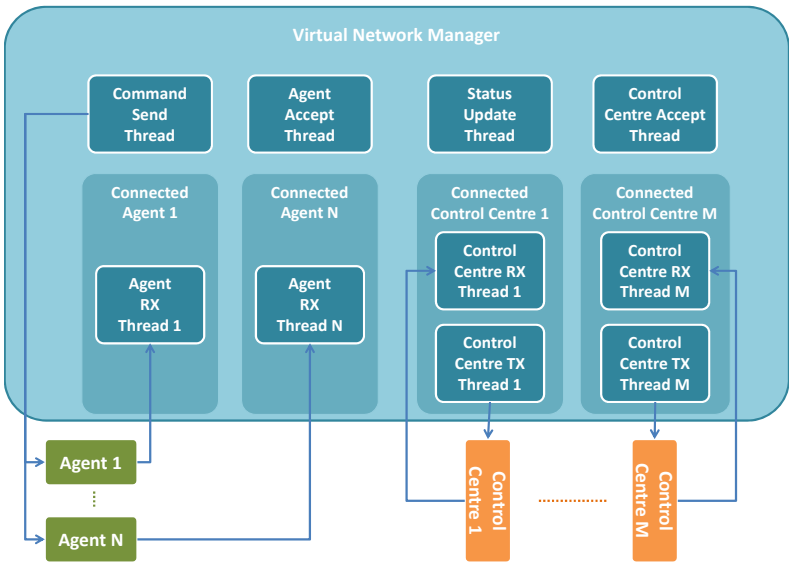


Fig. 5. Manager Module Decomposition.

3.3 Agent Module

The Agent is a module written in C that was designed to work within the *dom0* of a XEN [15] virtualization environment. Due to the multitude of tasks that it performs, there are several threads running within the module (Figure 6). They can be grouped into three main categories, according to their purpose.

The first group is related with the Manager communications. The *Manager Connection Thread* makes sure that a connection to the Manager exists, and is in charge of reconnecting in case of a connection failure. This thread is also responsible for signaling the remaining threads if the Manager connection is lost or established.

The *Keep Alive ID Request Thread*, besides acquiring a valid ID from the Manager at startup, periodically sends Hello beacons to the Manager to increase the connection robustness and allow the Manager to identify problematic

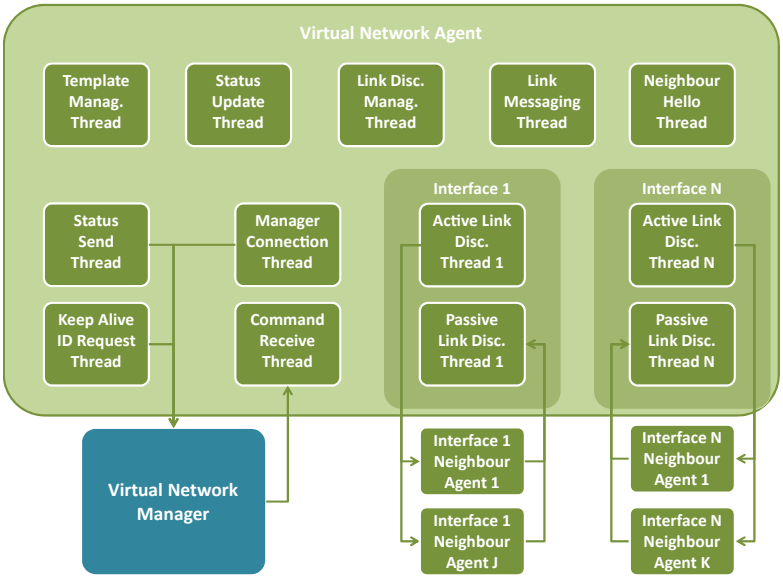


Fig. 6. Agent Module Decomposition.

connection situations. The *Status Send Thread* is the module's general purpose message gateway, since every message is sent to the Manager through it, except for the ID request and Hello beacons. The last thread in the Manager communications group is the *Command Receive Thread*, which is in charge of receiving the Manager's messages and process them accordingly.

The second functional group, which takes care of the physical node's state, is composed of two threads. The first one is the *Status Update Thread*, that periodically monitors the node for changes in configurations, load and state; while the second one, the *Template Management Thread* is responsible for the maintenance of a virtual machine pool, that may contain several templates, used for speeding up the node creation process. Both threads rely heavily on the *libvirt* API [16], for gathering information and performing hypervisor calls.

The last group is associated with the distributed discovery process. One *Active* and one *Passive Link Discovery Thread* exists for each active physical interface that handles the multicast communications on that interface. The *Neighbour Hello Thread* periodically sends multicast Hello beacons through the active interfaces' communication threads, while the *Link Discovery Management Thread* is responsible for managing the Active and Passive threads, i.e. launching or terminating them if a new interface becomes active, inactive or changes its configuration. The last thread in this group is the *Link Messaging Thread*, which is a message gateway between the Status Update Thread and the Active Discovery threads that is used to signal new and deleted local virtual resources.

4 Supported Scenarios

4.1 Discovering a Virtual Network

Through the algorithm described in 2.3, the Manager is aware of the complete physical and virtual topologies. Using the Control Center, it is possible to request information about a running virtual network through the *Get VNet* menu, where a list of available networks will be shown. By selecting a physical or virtual network, a new tab on the GUI will appear, showing information about the requested network. As can be seen on Figure 7, information about the network's resources, such as nodes, links, and their interconnections is shown.

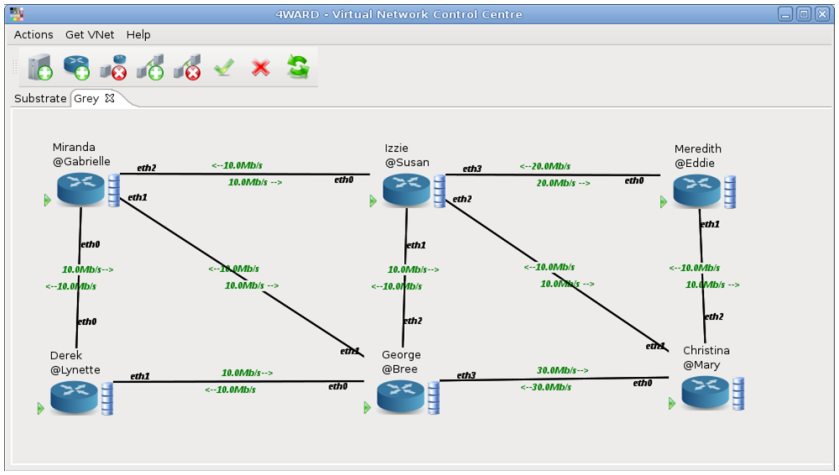


Fig. 7. Virtual Network Discovery.

4.2 Monitoring a Virtual Network

The discovery procedure is dynamic in nature, and therefore, the information displayed on the drawing canvas, belonging to a given virtual network, is also dynamic. The on-screen information will be updated in a close to real-time fashion. When watching a virtual network on the Control Centre, it is possible to see dynamic state, configuration, and load information. If, for example, a link goes down or suffers any change, it is possible to see it on the Control Centre. If a node becomes overloaded, the graphical icon representing the CPU load will change its color, thus warning the network administrator of a high load situation. Figure 8 exhibits some example monitoring information.

4.3 Creating a Virtual Network

The creation of a new virtual network is performed through the Control Centre. This can be done by loading a VNet specification Extensible Markup Language (XML) [4], on the *Load XML* menu, or by the *New VNet* option on the main menu. Through the New VNet option, after specifying a VNet name, a new tab will appear in the drawing canvas. In this canvas, it is possible to insert new virtual nodes, using drag-and-drop mechanisms, and interconnect them with links.

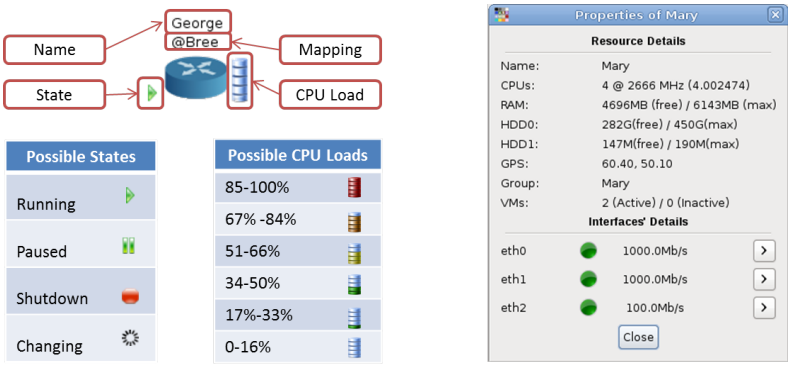


Fig. 8. Virtual Network Monitoring.

After placing the required resources, it is possible to configure their specifications, such as the number of required CPUs, RAM amount, number of interfaces, hard drive space and location requirements.

The virtual interfaces may be configured with IPv4 or IPv6 data, so that initial connectivity exists between the virtual nodes. As for the links, it is possible to specify their required bandwidth, latency, jitter, loss and associated virtual interfaces. Despite all these options, the QoS information is not currently enforced. Some examples of configuration windows are shown in Figure 9.

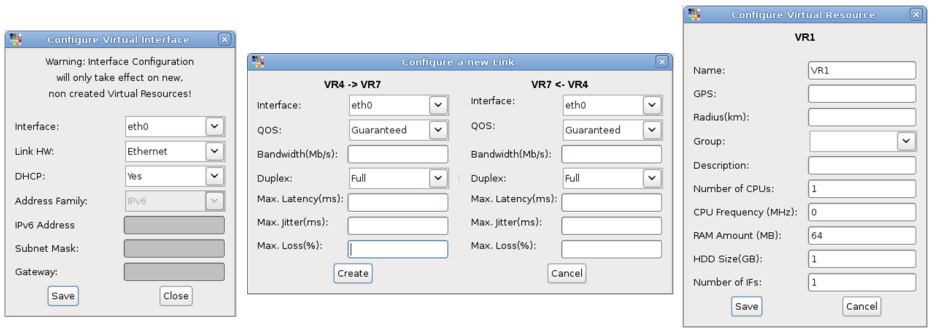


Fig. 9. Virtual Network Configuration Windows.

As soon as the nodes, interfaces and links are properly configured, the designed virtual network may be saved to an XML file, for future use, or be committed to the Manager, which will then deal with the embedding process. In the case of a successful mapping, the virtual nodes and links will be created and configured. As soon as the process terminates, the recently created virtual network will be displayed on the Control Centre, where the administrator will be able to verify the configurations and make sure that everything was created as desired.

4.4 Managing a Virtual Network

One part of the network's administrator tasks is to manage the existing virtual networks. This platform provides a basic set of management features, through the

use of the GUI. The access to the management features, available only to virtual resources, is performed through a menu when right clicking on the resource icon.

A resource state may be changed using the following commands: *Start*; *Shutdown*; *Suspend*; *Resume*; and *Destroy*. Besides changing the resource's state, it is also possible to change the RAM amount assigned to a virtual node in real-time.

5 Experimental Results

5.1 Testbed Description & General Assumptions

The experimental testbed is composed of 6 physical nodes, whose CPU and RAM characteristics are described in table 1. The nodes are interconnected with a total of 8 ethernet links, ranging from 100Mbps to 1Gbps.

Node	**Susan**	**Lynette**	**Gabrielle**	**Bree**	**Eddie**	**Mary**
CPU	PentiumD 950	PentiumD 950	Core 2 Duo E6400	Xeon E3110	Xeon X3220	Xeon X3330
RAM	6GB	6GB	4GB	6GB	6GB	6GB

Table 1. Testbed specification.

In the following tests, the Manager was running on a separate physical machine directly connected to the testbed (Intel Core 2 Duo P8600; 4GB of RAM; 100Mbps link). The created virtual networks were always a replica of the underlying physical network, and served the purpose of being a reference throughout the tests. The virtual nodes were configured with 1 CPU, 64MB of RAM, 1GB of HDD, and 1Mbps links. During the tests, all virtual nodes were idle and so were the physical nodes and links.

The maximum amount of created virtual networks was 40, which corresponds to 40 virtual nodes in each physical node. The results presented on the following sections always assume a 95% confidence interval.

5.2 Data Gathering

Data gathering is a very important feature, in order to have an updated view of the existing networks' status and characteristics. Its performance may be a critical factor: if the data gathering procedures takes too long, the reaction to failures or other events may be delayed.

Cold boot is described as the time it takes for each physical node to fully discover and update the information about itself and its virtual nodes, and send a full update to the Manager. This test intends to demonstrate the dependency between this start up time, the number of running virtual machines and the capability of the physical nodes. In order to assess the cold boot time, the time difference between the Agent start-up and the end of the first status update was measured. This procedure was repeated 10 times for every considered amount of virtual networks.

Figure 10 exhibits the time required to boot with the increase in the number of existing virtual machines. It is clear that the substrate nodes have very different capabilities, and that the boot time is heavily dependent on the CPU

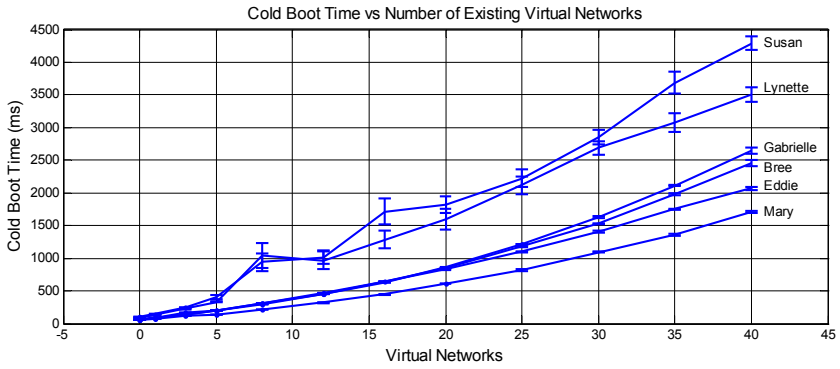


Fig. 10. Agent Cold boot results.

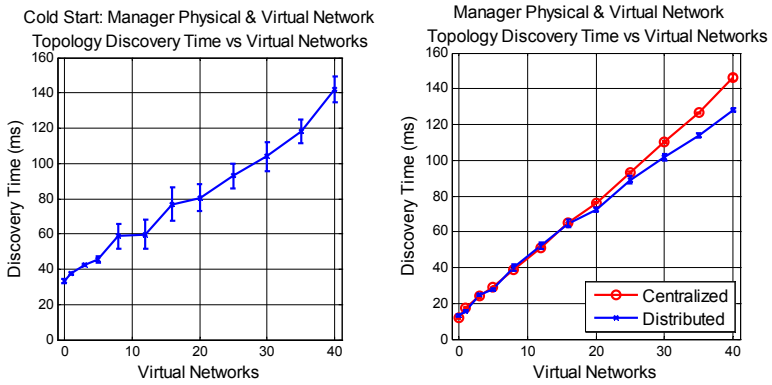
processing power. The physical nodes using the old Intel NetBurst architecture, i.e. Susan and Lynette, performed worse than the ones relying on the more recent Intel Core 2 architecture.

5.3 Network Discovery

In this subsection, two different tests were performed. The first one was the *Cold Network Discovery* test that measured the time elapsed since every Agent had booted up, with no neighbourhood knowledge, until the full physical and virtual network's topologies had been built by the Manager. This test took into consideration the time required for the Agents to gather neighbourhood data and for the Manager to attain a converged view of every topology.

The second test, the *Hot Network Discovery* scenario, measured the time that it took for the Manager to gather the network information from every Agent and build all network topologies, in the situation where the Agents already had the neighbourhood knowledge. This test aimed to assess the computing workload required by the topology building algorithm using the data attained through the distributed discovery, by comparing it to a centralized version of the discovery algorithm that did not rely on the neighbourhood data from the Agents, but instead relied on the configuration data of each physical and virtual node.

In the first test, for every virtual network created, the cold discovery time was measured 10 times. Each time, the Manager and Agents were firstly shutdown. Afterwards, every Agent was brought up. When every Agent had finished its cold boot, the Manager was started and began waiting until a pre-determined amount of nodes and links were received, depending on the number of the currently running virtual networks. As for the second test, the Manager was programmed to terminate its execution upon receiving and processing the expected amount of nodes and links, which was variable according to the number of virtual networks running at a given instant. It had two operation modes: the first one used a centralized topology discovery algorithm, while the second one used the link information sent by the Agents to the Manager to build the topologies. A script was created that executed the Manager 100 times in a successive way, both for the centralized and the distributed algorithms, with a 1 second delay between Manager termination and restart.



(a) Cold Network Discovery (b) Hot Network Discovery

Fig. 11. Virtual Network Cold and Hot Discovery

By observing Figure 11(a), it is possible to notice that the time required for discovery follows a linear trend. Taking into account the results obtained from the simulation results of the discovery algorithm [12], this trend was to be expected. Comparing the hot discovery results, displayed in Figure 11(b), with the discovery times of the previous subsection, it is possible to state that the discovery process is faster. This is to be expected since these measurements simply incorporate the time required for the Agents to send their neighbour information to the Manager, and the time it takes for the Manager to process or aggregate this information.

By comparing the results of the centralized and distributed approach, one can state that, as the number of virtual networks begins to grow, the distributed approach provides lower discovery times than the centralized one. For 40 virtual networks, the time difference is 20ms, or about 17%. This is a very important conclusion, as it shows that distributed approaches need to be supported in future, and more complex, networks.

The scalability of the distributed discovery algorithm is demonstrated both through these experimental results and the simulation results presented in[12].

5.4 Virtual Network Mapping

In this test, the performance of the proposed mapping algorithm was evaluated. In spite of the small-scale testbed, some insight should be gained about the scaling of the algorithm with the increase in the number of existing virtual networks. In order to assess the mapping times, 40 virtual networks, like the ones specified in 5.1 were created, one at the time. The time required for the Manager to process the received unmapped XML and return a mapped one was measured. The tests were repeated 3 times.

The time required to perform the mapping is shown to increase with the number of existing virtual networks (Figure 12). Since the mapping procedure only depends on the virtual network to be embedded and on the physical network,

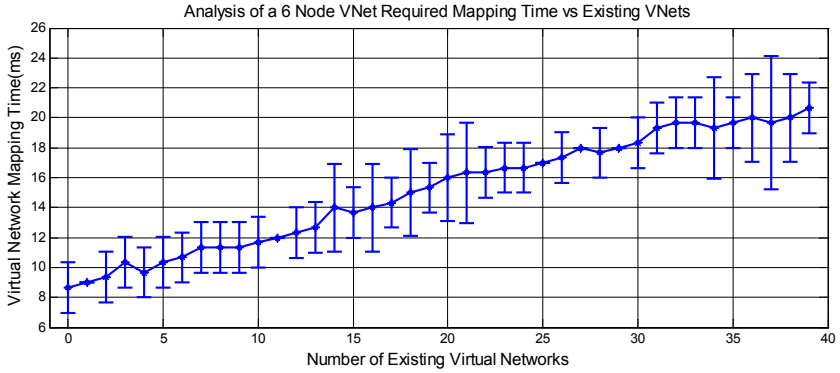


Fig. 12. Virtual Network Mapping results.

it would be expected that the mapping times remained constant. However, this was not the case. In order to understand the increase in the required mapping time, one must take into consideration that, when performing the mapping, the Manager needs to update the physical links' load, and therefore needs to access each existing virtual network's information. Thus, for each additional virtual network, the Manager will need more time to calculate the physical links' stress. This increment in needed time is revealed in the obtained results, which show a linear scaling with the number of existing virtual networks.

Regarding the absolute mapping times, they remained in the order of low tens of millisecond, which is very good and can be considered real-time. The considerable deviations on the measured mapping times are probably due to the Manager's need to lock the different resources' mutexes, while performing the mapping.

5.5 Virtual Network Creation

In order to evaluate the time required to create a virtual network on the available testbed and its scaling with the amount of previously existing virtual networks, several creation tests were performed.

The amount of previously existing virtual networks was varied between 0, i.e. without virtual networks, and 39. For each considered point, a virtual network was created and deleted 10 times, and the time required for creation was recorded. The created virtual networks were the same as the ones previously specified in 5.1. The considered creation time encompasses the time required for the Manager to split the mapped XML and send the different command messages to the Agents, as well as the time required for the Agents to report back with updated information about the created resources and links. The Manager was in charge of measuring these creation times.

The obtained results, shown in Figure 13, follow a linear trend with the increase in the amount of existing virtual networks. It is worth noting that the total creation time, encompassing both node creation and subsequent topology discovery, only depends on the slowest physical node, from the ones chosen to have a virtual node embedded. Considering the physical node's performance

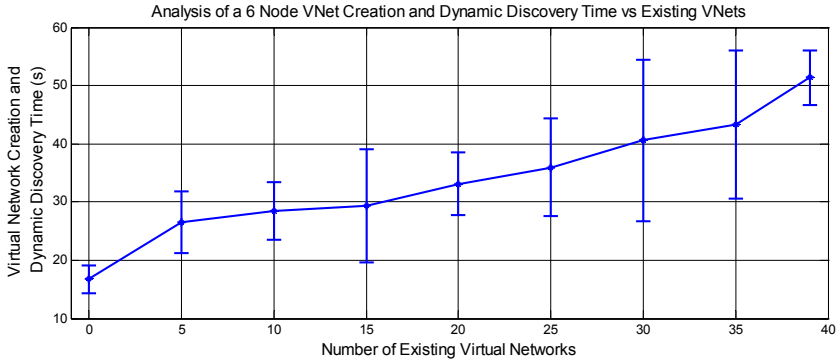


Fig. 13. Virtual Network Creation results.

estimates attained in subsection 5.2, one can see that the slowest node, Susan, is about three times slower than the fastest node, Mary.

The demonstrated increase in discovery times is due to the increase in time required to gather resource information. It is worth noting that, when the virtual node is created, the used virtual machine template will be regenerated, imposing a severe strain on the physical node's slow HDD, thus further slowing down the data gathering and subsequent discovery process.

Significant deviations were observed when measuring the creation time. Since these measurements depend on every physical node (their respective discovery threads, mutex locks, time required for gathering resource information and also the performance of the hypercalls), large variations were verified.

6 Conclusions

This paper presented the NVSS, which was designed to test the support of network virtualization. The developed platform provides functionalities such as discovery, mapping, monitoring and management of virtual networks and aggregates them in a easy to use user interface.

The results of the conducted tests show the performance of the platform on multiple situations. The scaling of the Agents' boot time was analysed, and different scenarios were considered on the assessment of the network discovery functionality performance. As for the embedding functionality, both the time required for mapping a new virtual network and creating the virtual network were studied. Considering the attained Agent boot results, they were heavily dependent on the physical's node performance and on the number of active virtual machines. With regard to network discovery times, in both scenarios, hot and cold discovery, they scaled linearly with the increase in the number of virtual networks. For the hot discovery scenario, where the distributed discovery algorithm was compared with a centralized version, the performance gains of the distributed approach were significant, especially for more than 20 virtual networks. The virtual network mapping and creation results showed a linear behavior. The heuristic algorithm was able to map new virtual networks with a computational time of some tens of milliseconds. In the virtual network creation

process, the results show that this procedure is still able to provide fast virtual network creation times even with 40 virtual networks embedded.

As future work, we plan to address the reconfiguration and migration features as a result of fault-tolerance mechanisms, and the inclusion of virtualization of both the network and cloud resources.

7 Acknowledgments

The authors are thankful to the members of the 4WARD project, particularly those involved in Work Package 3, for the collaboration and fruitful discussions.

References

1. Carapinha, J. and Jimenez, J., Network virtualization: a view from the bottom, in Proceedings of the 1st ACM workshop on Virtualized infrastructure systems and architectures. Barcelona, Spain: ACM, 2009, pp. 7380.
2. Architecture Conceptual Design for New Generation Network. http://akari-project.nict.go.jp/.
3. GENI: GENI - Global Environment for Network Innovations. http://www.geni.net/.
4. 4WARD Consortium: Virtualisation approach: Evaluation and integration - update. Technical report, ICT-4WARD project, Deliverable D3.2.1, June 2010.
5. Melo, M., Carapinha, J. and Sargento, S., Network Virtualisation from an Operator Perspective, Proc Conf. sobre Redes de Computadores - CRC, October, 2009.
6. Chowdhury, N. M. K. and Boutaba, R., Network virtualization: State of the art and research challenges, IEEE Communications Magazine, no. 7, July, 2009.
7. Norbert, E., Greenhalgh, A., Handley, M., Hoerdt, M., Huici, F., Mathy, L., and Papadimitriou, P.: The virtual router project. http://nrg.cs.ucl.ac.uk/vrouter/.
8. XORP: XORP - eXtensible Open Router Platform. http://www.xorp.org/.
9. Nogueira, J., Melo, M., Carapinha, J. and Sargento, S., Virtual Network Mapping into Heterogeneous Substrate Networks, submitted to IEEE Symposium on Computers and Communications (ISCC) 2011, June 2011.
10. Zhu, Y. and M. Ammar: Algorithms for assigning substrate network resources to virtual network components. In INFOCOM 2006. 25th IEEE International Conference on Computer Communications. Proceedings, pages 112, 2006.
11. Lischka, J. and Holger, K.: A virtual network mapping algorithm based on subgraph isomorphism detection. In VISA '09: Proceedings of the 1st ACM workshop on Virtualized infrastructure systems and architectures, pages 8188, New York, NY, USA, 2009. ACM, ISBN 978-1-60558-595-6.
12. Nogueira, J., Melo, M., Carapinha, J. and Sargento, S., A Distributed Approach for Virtual Network Discovery, IEEE Globecom 2010 Workshop on Network of the Future, December 2010.
13. PlanetLab: PlanetLab - An Open Platform for Developing, Deploying, and Accessing Planetary-Scale Services. http://www.planet-lab.org/.
14. A. Udugama, L. Zhao, Y. Zaki, C. Goerg, A. Timm-Giel, End-to-end Performance Evaluation of Virtual Networks using a Prototype Implementation, Second International ICST Conference on Mobile Networks and Management (MONAMI), Santander (Spain), September 2010.
15. Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I. and Warfield, A.: Xen and the art of virtualization. SIGOPS Oper. Syst. Rev., 37:164177, October 2003, ISSN 0163-5980.
16. libvirt - Virtualization API - http://libvirt.org/