Autonomic Communications & Network Virtualization: A Real Self-organizing Model for Substrate Networks

Clarissa Marquezan, Giorgio Nunzi, Marcus Brunner, Lisandro Granville

***Abstract*—In this paper we propose the joint use of autonomic communications and network virtualization. We defined a full decentralized self-organizing model which aims to reduce the overall traffic of the substrate network by migrating virtual nodes. The specific contributions of this work are: (i) definition a decentralized management architecture for network virtualization; (ii) heuristics to identify cut-through traffic pattern overloading links of the substrate network only correlating information of local resources; (iii) and a a fully decentralized self-aware, and self-configuring algorithm to negotiate the migration of virtual resources.**

***Index Terms*—Self-organizing, Network Virtualization, Resource Usage, Maitenance, Virtual Resource Migration**

I. Introduction

AUTONOMIC communications is a suitable approach to deal with complex and dynamic networks [1]. The key of autonomic communications is of building sophisticated networks capable to manage themselves and deal with changes in the environment. Among several scenarios employing autonomic communications [2][3][4], virtual environments deserve special attention due to their complexity, dynamics and potential to be economically explored. Network virtualization [5][6][7], for instance, is emerging as a promising technology able to improve the large deployment of networks and the economical relationship between service providers and substrate network providers.

In fact the benefit of network virtualization is to outsource the operational costs associated to the physical infrastructure to a single provider. For instance, multimedia providers may deploy their services, like IPTV services, without dealing with high investments on the physical infrastructure along different networks (core, metropolitan, and home networks, for example) [8]. Efficient use of resources becomes of paramount importance to operate the substrate network in network virtualization context. The complex dependencies and dynamic changes on requirements of resources [9][10] demand sophisticated management techniques, like autonomic communications.

Manuscript received December 1, 2008; revised January 11, 2007.

Clarissa Marquezan is Phd candidate of Institute of Informatics, Federal Univertisity Of Rio Grande do Sul, Porto Alegre, Brasil, and Phd intern student at NEC Network Research Laboratories, Heidelberg, Germany (e-mail: clarissa@inf.ufrgs.br).

Giorgo Nunzi and Marcus Brunner are with NEC Network Research Laboratories, Heidelberg, Germany (e-mail: {nunzi, brunner}@nw.neclab.eu).

Lisandro Granville is with Institute of Informatics, Federal Univertisity Of Rio Grande do Sul, Porto Alegre, Brasil (e-mail: granville@inf.ufrgs.br).

Indeed, the autonomic solutions proposed so far for virtual environments present limitations to maintain the efficient use of the resources in the presence of dynamic, transient changes because they use a centralized, total-view, and off-line approach. The major limitations are the low responsiveness to network changes, the overhead introduced by the management traffic going until the central entity, and the high latency of analysis and enforcement of reallocation process. Taking into account that network virtualization is in its infancy [9], the maintenance of efficient resource consumption on the substrate network under dynamic, transient conditions is, to the authors knowledge, a rather untouched research topic. Thus, we propose in this paper the joint use of autonomic communications and network virtualization to efficiently maintain the use of substrate resources.

We defined a full decentralized self-organizing model which goal is to reduce the overall traffic of the substrate network by migrating virtual nodes. The algorithms of the self-organizing model are executed by each physical node of the substrate network, dismissing any kind of central entity. The analysis of the conditions that triggers the resource reallocation uses only local information of the physical node where the analysis is executed. The adoption of a fully decentralized execution supported by localized information, allows each management entity to make online and autonomous decisions. The parallel execution of all management entities produces a self-organizing substrate network. In practical terms, the contributions of this work are:

- the definition of a decentralized management architecture for network virtualization;
- heuristics to identify a cut-through traffic pattern overloading links of the substrate network only correlating information of local resources;
- and the definition of a fully decentralized self-aware, and self-configuring algorithm to negotiate the migration of virtual resources.

To validate the self-organizing model proposed here, we developed an Omnet ++ module for the decentralized control and management architecture of the virtual model considered in this paper. We implemented the self-organizing algorithms to manage the substrate network of this virtual model. The evaluation scenario is based on IPTV services. As mentioned before, the deployment of IPTV services involves heavy investments on the physical infrastructure. So far, IPTV providers are good candidates to use network virtualization techniques to

deploy their resources in a virtual network. Moreover, the IPTV traffic can dynamically change according to the user's preferences for a certain content, and this change is exactly the object of study in our model. Being this, we simulated virtual IPTV networks. We compared the total traffic of the substrate network (with and without the self-organizing model) under varying user's request rates for each virtual IPTV network. The results show the efficiency of our self-organizing model and its viability to self-manage the use of substrate resources in network virtualization.

The remainder of this paper is described as follows. The management architecture of the network virtualization mode considered in this work is presented on Section 2. We introduce the self-organizing model on Section 3. The evaluated scenario is described on Section 4, while results associated to this evaluation are discussed on Section 5. The related work is presented in Section 6. Finally, Section 7 brings the remaining challenges and the conclusions of this work.

II. Network Virtualization Model

As first step, we illustrate the reference architecture for virtual networks that we adopt in our work as a model. A virtual network replicates entirely the network elements commonly present in a physical infrastructure, like nodes and links. Nevertheless, the coexistence of two different strata (virtual and physical) and their mapping needs to be properly characterized; we do this, with the help of Fig. 1. The substrate network comprehends all physical resources belonging to an infrastructure provider, and the virtual networks (VN) are slices of the substrate network resources assigned to virtual providers. Virtual links are connections between two adjacent virtual nodes of the same virtual network. As illustrate in Fig. 1, virtual links can have a one to one mapping to substrate links (e.g., the link A#1-C#1 of VN#1), or span different links on the substrate network, generating a cut-through traffic on some substrate nodes (e.g., the link A#1-E#1 of VN#1 creates a cut-through traffic on the physical Node B).

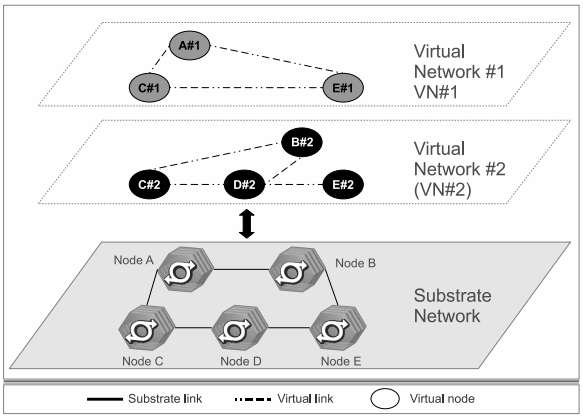

Fig. 1. Network virtualization environment

It is out of the scope of this paper to characterize the economical model, the parameters, and algorithms required to define and map virtual networks into the substrate network. Further details about the network virtualization can be found in specific projects like GENI [11] and 4WARD [12], or in [9][10]. However, some minimal assumptions are necessary and presented in Fig. 2.

A *Virtual Node* is composed of a set of assigned fraction of resources of the substrate node, like CPU clocks, memory, link bandwidth, and a certain storage capacity. An important aspect in the architecture of virtual nodes is the transparency. Virtual nodes from different virtual networks cannot see or exchange any type of information to assure isolation among the providers. Additionally, the data exchanged in the virtual node is transparent to the substrate node to preserve the privacy of the virtual customers. Nevertheless, some minimal primitives to inspect the activity of different slices are normally available: as an example, the substrate controllers are normally allowed to read the amount of computational resources used (e.g., CPU, memory or disk) and traffic consumed.

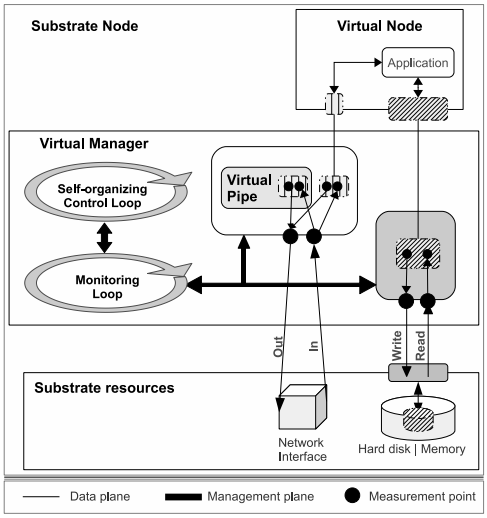

Fig. 2. Substrate node architecture

The allocation and supervisions of the slices of a Virtual Node is performed by a dedicated element, that we call *Virtual Manager*. In an advanced setting with autonomic capability, the *Virtual Manager* is also the element executing the self-organizing algorithms. As presented in Fig 2, the *Monitoring Loop* gathers relevant measured data for the self-organizing control loop: the relevant information is the amount of network traffic (the white box on the *Virtual Manager* module) and reads/writes of memory of each virtual node deployed on the substrate node (the light gray box on the *Virtual Manager* module).

The resource consumption of a substrate node can be divided into two classes; (i) resources allocated to a Virtual Node and (ii) the cut-through traffic between Virtual Nodes executed on different substrate nodes, like illustrated in Fig. 1. We believe that the identification of the source consumer of resources of a substrate node is essential for deciding how to optimize resource allocation. Therefore, we define as *Virtual Pipe* the cut-through traffic inside a substrate node. From the point of view of the Virtual Manager, the Virtual Pipe and the Virtual Node are treated equally as consumer of resources, but the first one is not visible in the virtual network topology.

Note, that the virtaul pipe concept can be implemneted in different ways. The key though for our work is, that the Virtual Managager is able to measure the traffic of the cut-through traffic and is able to associate it with a Virtual Network.

The *Self-organizing Control Loop* executes the autonomic algorithm in every substrate node. Its purpose is to change or optimize the traffic by moving a virtual node associated to this traffic to the substrate node. Our approach relies on a completely distributed execution of the optimization algorithms, where the evaluation and decision to reallocate resources is performed on each node in coordination with a substrate neighbor node. The next section describes the details of the self-organizing system proposed in this paper.

III. Self-organizing Model

The self-organizing model is characterized by the use of local information to identify bottlenecks situations, and by the full distributed decision-making process to reallocate the virtual resources. In this section, we show the virtual manager roles and how they are related to the heuristics defined to characterize traffic patterns. We also show the heuristics themselves, and the self-organizing control loop.

A virtual manager is designed to be an autonomous entity that makes decisions to reduce the resource consumption based only on the information monitored on its substrate node. So, when a link of a substrate node is identified as overload the virtual manager triggers the process to identify the traffic pattern of each virtual network using the resources of this substrate node. The goal is to identify if some of the virtual networks is presenting the pattern associated with cut-through traffic.

We named the pattern associated with cut-through traffic as "forward traffic". The forward traffic is described as a flow departing from a virtual node, passing-by virtual pipes (or even virtual nodes) of the same virtual network, and arriving in a distinct virtual node. Fig.3 shows examples of forward traffic on virtual networks #1 and #2 (respectively VN#1 and VN#2). In the case of VN#1, the flow departs from substrate node A, passes through the virtual pipe on substrate node B and is forwarded until the destination. Considering the case of VN#2 the flow departs from substrate node C, passes-by the virtual pipe on substrate node A, and reaches the destination on substrate node B.

Let's suppose that links l_1^s and l_2^s (both belonging to L^s - notation on Table I) are considered overloaded by the virtual managers of substrate nodes C/A, and A/B, respectively. Link l_1^s has only the traffic flow from VN#2, while link l_2^s contains traffic flows from both virtual networks. The challenge of the self-organizing algorithm running within the virtual managers is to "guess" if the flows inside these links match the forward traffic pattern without getting any extra information than the resources locally used by the virtual network.

We explicitly use the term "guess" because we do not inspect the content of the traffic running inside the virtual network. In contrast, we transparently account the amount of resources used by the virtual network without knowing the semantic associated to the accounted value. Thus, our traffic pattern identification is accomplished by heuristics defined to search for the forward pattern among the clues given by the resources consumed by each virtual network inside the substrate nodes.

The presence of a virtual network in a substrate node can be accomplished by distinct virtual elements (pipes or nodes), and the substrate resources associated to these elements are different. So, the guessing process is different for virtual pipes and for virtual nodes, and for this reason two heuristics were elaborated. The first one is called *receiving candidate* heuristic and investigates the resources used by a virtual pipe associated with the virtual network. The second one is named *moving candidate* heuristic and it identifies a forward traffic pattern when a virtual node of a virtual network is deployed on the substrate node. The details of these heuristics are explained in the sequence.

The output of the heuristics is a list of virtual networks to be received or moved. These lists are called, respectively, receiving and moving candidate. An example of this output is presented in Fig. 3. The self-organizing algorithms running in the substrate node C identified the virtual node of VN#2 as a moving candidate, while substrate nodes A and B identified virtual nodes from VN#1 and VN#2 as moving candidates, and virtual pipes from VN#2 and VN#1 as receiving candidates.

Based on the traffic pattern characterization, and the cost evaluation of reorganizing virtual resources, the virtual manager of each substrate node take the actions to reorganize the resources of the virtual networks identified in the candidate lists without a global view of the resources. Bellow, we present the details of the decision-making to reorganize the resources. The notation used in the following subsections is described in Table I, superscript denotes a virtual or a substrate element, and subscript is used to identify the indexes of elements.

B	Maximum bandwidth capacity of a link
T	Total amount of traffic
IN	Incoming traffic
OUT	Outgoing traffic
N_{vl}^{sl}	Number of virtual links inside the substrate link
L^s	List of links in a substrate node
VL^s	List of virtual links inside a substrate link
$VNETL^s$	List of substrate links of a virtual network in a substrate node
$OvLink^s$	List of overloaded links in a substrate node
$OvVLink\_l^s$	List of overloaded virtual links in a substrate link
$read^v$	Amount of reads of virtual node
$write^v$	Amount of writes of virtual node

TABLE I
Notations of Self-organizing Model

A. Identification of an Overloaded Substrate Link

In our model an overloaded link is the element that starts the evaluation of possible reorganization of virtual resources. So, for each link $l_i^s \in L^s$, where i is the index of L^s, the condition presented in (1) verifies if the substrate link l_i^s should be included in the list of overloaded substrate links $OvLink^s$.

$$l_i^s \in OvLink^s \text{ if } Tl_i^s > Bl_i^s * \alpha \quad (1)$$

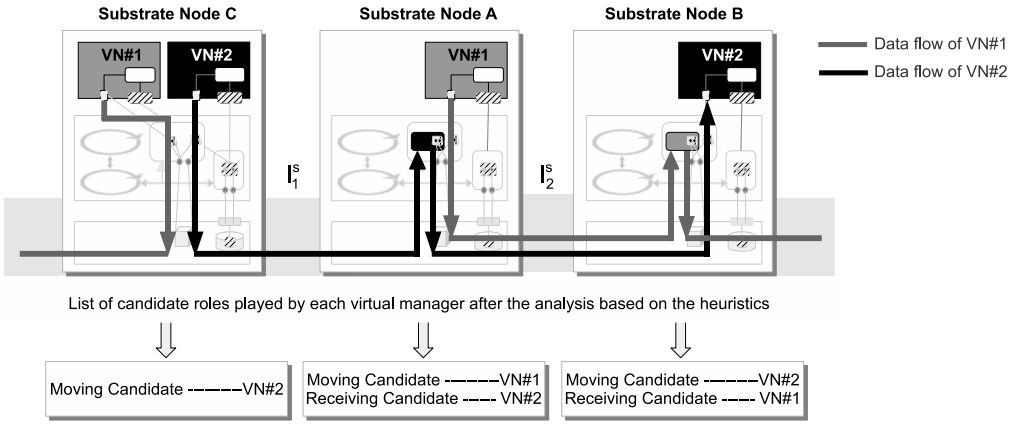

Fig. 3. Forward traffic pattern. The virtual networks presented in this figure are based on the virtual topologies used in Fig. 1

If the OvLink^s is not empty, the next step is to identify which virtual link inside each overloaded substrate link is consuming the resources. We defined the function *MajorT()* (2) applied for each substrate link $l^s_j \in$ OvLink^s, where j is the index of the substrate link inside the list OvLink^s. The goal is to verify which virtual link l^v_k is consuming the major part of the resources of the substrate link l^s_j, where k is the index of the virtual network. For each substrate link l^s_j a list OvVLink_l^s_j is created with the l^v_{kj}, where l^v_{kj} is the virtual link of virtual network k inside the substrate link l^s_j.

$$l^v_{kj} \in OvVLink\_l^s_j \text{ if } MajorT(l^s_j) = true, \text{ where}$$
$$MajorT(l^s_j) = \begin{cases} true & \text{if } Tl^v_{kj} \geq Tl^s_j - Tl^v_{kj}, \\ false & \text{otherwise} \end{cases} \quad (2)$$

After the identification of the overloaded virtual link starts the process to determine whether the flow(s) associated to the virtual link matches with the forward traffic pattern.

B. Receiving Candidate Heuristic

As described above, the receiving candidate heuristic is applied to identify a forward traffic when a virtual pipe is associated with the overloaded virtual link inside the link l^s_j. In a reduced virtual network topology like the one presented in Fig. 3 the identification of a forward traffic pattern in a virtual pipe is trivial, because the input in one virtual/substrate link is the output in the other virtual/substrate link. However, a complex topology, where a virtual pipe is connect to more than two virtual links, requires a detailed correlation of the traffic flowing through the virtual links of this virtual pipe. The receiving candidate heuristic elaborated in this work considers these complex topologies.

We defined a set of conditions related to the traffic passing through the virtual pipe that must be analyzed before declaring a substrate node as a receiving candidate. [1]. The analysis of the virtual pipe traffic can be done comparing the incoming traffic against the outgoing traffic of the analyzed virtual link, or vice-verse. For the receiving candidate heuristic we based the analysis on the comparison of the incoming traffic against the outgoing traffic. The conditions listed below are applied for each virtual link l^v_{kj} belonging to each OvVLink_l^s_j list.

- **Condition 1:** Guarantees that no read and write is associated to the virtual network *k*.
$$ForwardT = \begin{cases} 1 & \text{if } (read^v_k = 0) \wedge (write^v_k = 0) \\ 0 & \text{otherwise} \end{cases}$$
- **Condition 2:** Identify if the main traffic of l^v_{kj} is an incoming traffic.
$$IN\_MainT = \begin{cases} 1 & \text{if } INl^v_{kj} \geq Tl^v_{kj} - INl^v_{kj}, \\ 0 & \text{otherwise} \end{cases}$$
- **Condition 3:** Correlate the amount of incoming traffic of l^v_{kj} with single outgoing traffic of the same virtual network in other substrate link belonging to VNETL^s. This correlation enables the identification of the case that all traffic arriving in l^v_{kj} is entirely forwarded to a single virtual link.
$$SingleOut = \begin{cases} 1 & \text{if } \forall\ l^s_y \in VNETL^s_k : INl^v_{kj} \geq OUTl^v_{ky}, \\ 0 & \text{otherwise} \end{cases}$$
- **Condition 4:** Correlate the amount of incoming traffic of l^v_{kj} with all outgoing traffic of the same virtual network in other substrate links belonging to VNETL^s. The goal is to detect whether the incoming traffic on l^v_{kj} is forwarded to multiple distinct virtual links of the same virtual network.
$$MultipleOut = \begin{cases} 1 & \text{if } \forall\ l^s_y \in VNETL^s_k, \\ & \text{and } y \neq j : INl^v_{kj} \leq \left(\sum_y OUTl^v_{ky}\right) \\ 0 & \text{otherwise} \end{cases}$$

The final analysis presented in (3) is able to determine if the virtual link associated with a virtual network under analysis on the substrate link l^s_j is supposed to be inserted in the *Receiving_Candidate_List*. This list is a tuple <*virtual network, virtual link*>, where the first element of the tuple can have multiple entrances, but the second one is unique.

$$< vnet_k, l^v_{kj} > \in Receiving\_Candidate\_List \ if$$
$$ForwardT \wedge (IN\_MainT \vee (SingleOut \vee MultipleOut)) \quad (3)$$

The virtual manager of the substrate node will use the *Receiving_Candidate_List* during a cycle of the self-organizing control loop.

[1] Virtual nodes can behave like virtual pipes forwarding the traffic. However, this case requires different conditions which are out of the scope of this paper.

C. Moving Candidate Heuristic

The moving candidate heuristic complements the analysis of a forward traffic considering the perspective of a virtual node. We compare the incoming traffic against the outgoing traffic of each virtual link l^v_{kj} belonging to each *OvVLink_*l^s_j list. For the moving candidate heuristic we have to identify if a virtual node is generating the outgoing traffic inside l^v_{kj}.

In this work, we assume that a virtual node need to use other resources than just the virtual links to originate a traffic flow in l^v_{kj}. We also assume that the virtual node also uses the resources from storage devices, like memory or hard disk. For example, a virtual streaming server needs to read the requested media from some storage before sending this data to the requester. Thus, the moving candidate heuristic identifies a relationship between the outgoing traffic of link l^v_{kj} with the amount of *read*^v_k of the virtual network k. To establish this relationship a different set of conditions are required to identify the forward traffic pattern on a virtual node, and here as well, the conditions are applied for each virtual link l^v_{kj} belonging to *OvVLink_*l^s_j.

- **Condition 1:** The virtual network *k* must read data from its virtual storage slice.

$$ReadData = \begin{cases} 1 & \text{if } (read^v_k > 0) \\ 0 & \text{otherwise} \end{cases}$$

- **Condition 2:** The outgoing traffic of l^v_{kj} must be higher than the incoming traffic of the same virtual link.

$$OUT\_MainT = \begin{cases} 1 & \text{if } OUTl^v_{kj} \geq Tl^v_{kj} - OUTl^v_{kj}, \\ 0 & \text{otherwise} \end{cases}$$

- **Condition 3:** The outgoing traffic of l^v_{kj} must be associated with an amount of data retrieved from *read*^v_k). The problem here is to identify the amount of data read from the virtual storage and forwarded through l^v_{kj}, because we do not inspect any kind of data packet of the virtual network. So, the only way to identify the amount of reads flowing as outgoing traffic of l^v_{kj} is defining a similarity relation between the outgoing traffic of l^v_{kj} and the resources consumed by the virtual network *k*. So far, we defined a similarity function *Sim()* that determines if a given amount of *read*^v_k belongs to the "interval of similarity" of the outgoing traffic on l^v_{kj}.

$$Sim(l^v_{kj}, V) = \begin{cases} 1 & \text{if } SimBottom(l^v_{kj}) \leq V \leq SimUp(l^v_{kj}), \\ 0 & \text{otherwise} \end{cases}$$

where,
$SimBottom(l^v_{kj}) = l^v_{kj} - (l^v_{kj} * \delta)$,
$SimUp(l^v_{kj}) = l^v_{kj} + (l^v_{kj} * \delta)$, and
$V = \left(\left(\sum_y INl^v_{ky} \right) + read^v_k \right) - \left(\left(\sum_z OUTl^v_{kz} \right) + write^v_k \right)$,
where $\delta \in \Re : 0 \leq \delta \leq 1$,
y, z are the index of $VNETL^s_k$, and $z \neq j$.

The list *Moving_Candidate_List* is created after the analysis presented in (4), and this list is also composed of the tuple <*virtual network, virtual link*>.

$$< vnet_k, l^v_{kj} > \in Moving\_Candidate\_List \ \ if \\ ReadData \wedge OUT\_MainT \wedge Sim(l^v_{kj}, V) \quad (4)$$

In the sequence, we present the self-organizing control loop that analyzes the traffic of the substrate links according to the heuristics described above, and if necessary applies a moving mechanism to reallocate the virtual resources.

D. Self-organizing Control Loop

The self-organizing control loop is detailed in Algorithms 1, 2 and 3. The first stage of the control loop is to characterize the conditions that trigger the reallocation of the virtual resources. The equations previously described in this section are used to locally identify the status of the resources in a substrate node and create the receiving and moving candidate lists (between steps 1 and 4 of Algorithm 1). Atfer this first stage the Algorithms 2 and 3 are executed in parallel in the same substrate node, as stated in "Step 6" of Algorithm 1.

During the execution of Algorithms 2 and 3, an offer-based behavior is played by the virtual managers. The Algorithm 2 makes the virtual manager of a substrate node, communicate with its substrate neighbors offering itself to receive a virtual node associated with a virtual network presenting some cut-through traffic. The virtual manager offering itself does not know what are the virtual infrastructures of their neighbors. For this reason, the Algorithm 3 is designed to receive those offers/requests to move a virtual node and choose the best one that matches with its requirements to move a virtual node.

In case of a positive matching, there is also the evaluation of the cost-efficiency relation to execute the migration of the requested virtual node (Step 3 of Algorithm 3). An initial study about the cost-efficiency on migrating virtual nodes was described in [13]. If the matching and the evaluation are positive both algorithms will prepare the infrastructure required to apply the migrating mechanism. It is out of the scope of this paper describe the details of the migrating mechanism.

The key of Algorithm 2 is to create an ofter to a virtual node,

Algorithm 1 Self-organizing Control Loop

Step 1	Create *OvLink*^{s} list (based on Equation (1)).
Step 2	Create *OvVLink_*l^s_j for each l^s_j in *OvLink*^{s} (based on Equation (2)).
Step 3	For each link l^v_{kj} in *OvLink*^{s} verify if it belongs to *Receiving_Candidate_List* (based on Equation (3).
Step 4	For each link l^v_{kj} in *OvLink*^{s} verify if it belongs to *Moving_Candidate_List* (based on Equation (4).
Step 5	If *Receiving_Candidate_List* and *Moving_Candidate_List* are empty, GOTO **Step 7**.
Step 6	Execute **Algorithm 2** and **Algorithm 3**.
Step 7	Wait next self-organizing control loop cycle and GOTO **Step 1**.

Our self-organizing model considers that a virtual node can not migrate to a substrate node that is not adjacent. A multiple hop migration violates our assumption of local based decision-making, because the migration of a virtual node among

Algorithm 2 Receiving Candidate Algorithm

Step 1 For each $vnet_k$ in *Receiving_Candidate_List* find the most overloaded link l^v_{kj} and send request for the substrate neighbor node of this link to move the virtual node of $vnet_k$.

Step 2 For each request wait for the reply of the substrate neighbor during an interval of time *t*.

Step 3 If negative reply for $vnet_k$ remove the tuple associated with $vnet_k$ and l^v_{kj} from *Receiving_Candidate_List*, GOTO **Step 2**.

Step 4 If positive reply for $vnet_k$ activate migration under the perspective of substrate node that receives the virtual node.

Step 5 If timeout of interval t for $vnet_k$, remove all tuples with $vnet_k$ from *Receiving_Candidate_List*.

Algorithm 3 Moving Candidate Algorithm

Step 1 For each $vnet_k$ in *Moving_Candidate_List* wait an interval of time *t* for a request to move the virtual node associated with $vnet_k$.

Step 2 If there is a $vnet_k$ on *Moving_Candidate_List* that matches with the received request, GOTO **Step 3**, otherwise, send a negative reply to substrate node owner of the received request and GOTO **Step 1**.

Step 3 If the cost-efficiency is positive for $vnet_k$, send a positive reply message to substrate neighbor, remove all tuples with $vnet_k$ from *Moving_Candidate_List*, and activate migration under perspective of substrate node that moves the virtual node, otherwise send negative reply.

Step 4 If timeout of interval t for $vnet_k$, remove all tuples with $vnet_k$ from *Moving_Candidate_List*.

different substrate nodes requires the analysis of all resources involved in the process. We can ensure that a multiple hop migration will never happen through the employment of the negative answer on Step 2 of Algorithm 3 that avoids a request to move a virtual node be forwarded among neighbors until find a positive match.

In this sense, if there is a forward traffic between two virtual nodes separated by a chain of virtual pipes, the elimination of the virtual pipes (i.e., the cut-through traffic), and thus the approximation of the virtual nodes will require more than one self-organizing cycle. However, given the nature of the algorithms running inside the self-organizing control loop, both virtual nodes can migrate in parallel during the same cycle.

IV. TESTBED

An architecture proposed for IPTV [14] is depicted in Figure 4. The Super Head-End (SHE) element of such architecture is responsible for receiving and storing the flows, in this case TV channels and videos, from national content providers. These flows are forwarded through a core network infrastructure, and stored on the Video Hub Offices (VHO). Then the flows are transported over the metropolitan network and stored on the Video Serving Office (VSO) devices. Finally, the IPTV content reaches the home network and is delivered to the Set-Top-Box inside the television of the end user. For this testbed we assume that the SHE, VHO, and VSO are the elements virtualized and we show the self-organizing model applied to manage the VSO elements.

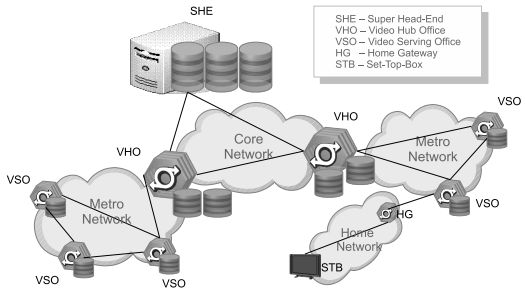

Fig. 4. IPTV Architecture

The testbed is composed of two IPTV providers covering the same geographic region. The network topologies (substrate and virtual) and the initial mapping of the virtual IPTV networks are depicted in Fig. 5. We consider a ring topology for both substrate and virtual networks, since this kind of topology is presented as a economically alternative for the deployment of IPTV infrastructures [8]. The substrate network is composed of 9 substrate nodes ("sn"), and each virtual IPTV network is composed of 3 adjacent nodes on the virtual network topology, but not directly connected on the substrate network topology. Between each virtual node there are 2 virtual pipes[2]. To simulate the traffic inside the virtual networks, we defined a user request model (extending concepts of [15]). The next subsections present the user's request model, the routing process within the virtual networks, and the monitoring process to collect data used by the self-organizing control loop.

A. User's Request model

There are two main concepts on our request model: community and preferences. A community characterizes the behavior of a group of users belonging to a virtual IPTV provider. There is a direct mapping between a community and a virtual network. The main parameters of a community are: number of movies requested per hour - request rate λ (reqs/hour), and the preference associated with each movie. A preference determines how often a movie will be requested, and it is defined as an array of ranges. For example, a community#1 has 3 movies and the array of ranges describing the preferences of the movies contains the values "0.3, 0.7, 1". This means

[2]The number of virtual nodes and virtual pipes can be defined by a cost model relating the amount of resources used for deploying a virtual network and the flexibility desired to reorganize the substrate network. This cost model is not the focus of this paper.

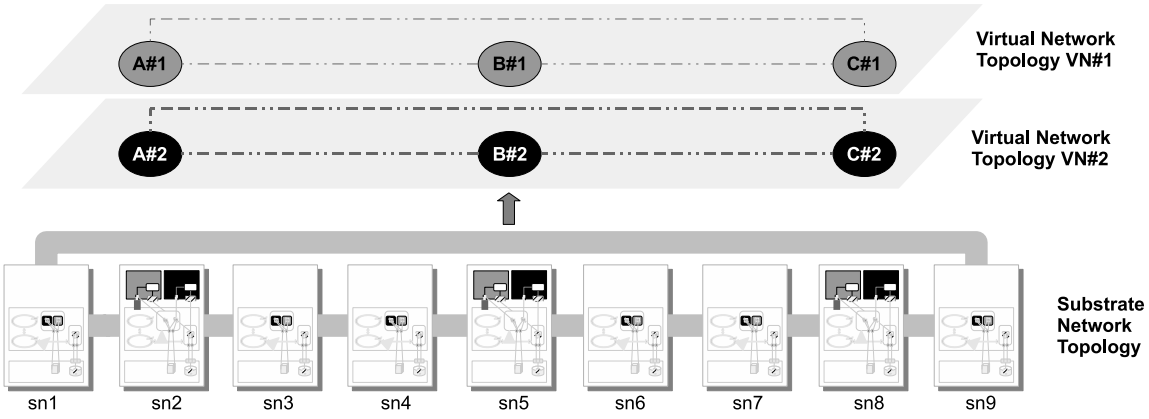

Fig. 5. Virtual topologies and initial mapping of virtual elements

that movie#1 has 30% of changes to be chosen, movie#2 40% and movie#3 30%.

A request generated in our model is denoted by the tuple <*comm_id, req_time, vnode_src, movie*>. The *comm_id* parameter is determined by a round robin selection on the list of communities. The time for the next request follows the exponential distribution presented in (5).

$$req\_time \sim Exp\left(\frac{1}{(\lambda/3600)}\right) \tag{5}$$

The source virtual node and the requested movie are required by the IPTV applications running within the virtual networks, so that a data flow can be created. The virtual node requesting a movie is chosen according a uniform distribution among all virtual nodes of its virtual network ($U(low\_vnode\_id, high\_vnode\_id)$). The requested movie is defined based on function $DefMovie(comm\_id, uniform)$, where $uniform \sim U(0,1)$. The random value $uniform$ will be compared with the interval $Range$, a field of list $MapRange$ (that maps the range array of preference for each movie of a community). For instance, following the example of the array of ranges described above, for community#1 and movie#1, $Range$ has the interval $[0, 0.3)$, for movie#2 $Range$ is equal $[0.3, 0.7)$, and movie#3 has $Range$ equal $[0.7, 1)$. The definition of the movie is detailed in (6).

$$DefMovie(comm\_id, uniform) = \begin{cases} movie & \text{if } \exists < comm, Range, movie > \in MapRange : \\ & (comm\_id = comm) \wedge (uniform \subset Range) \\ -1 & \text{otherwise} \end{cases} \tag{6}$$

The process to generate a user request finishes with the definition of the virtual node hosting the selected movie. Being this, the request is sent to the destination node, that identifies the source virtual node and start the transmission of the requested movie. The load of the virtual networks will vary according with the request rate and the movie preference. The request rate impacts the total amount of traffic consumed by the virtual network, and the preference determine which links will be more overloaded by the requests of the users.

B. Routing Process Within the Virtual Networks

To enable the packet transmission between virtual nodes we use a shortest path routing algorithm provided by the simulator Omnet++. The routes to the virtual nodes are recalculated periodically during the life time of a virtual network. The routing algorithm receives "hints" from virtual manager of the substrate node, about the presence of virtual pipes on substrate neighbors. Thus, a virtual node can find a path to a virtual neighbor even if there is no substrate adjacent connection between the virtual neighbors.

C. Monitoring Process

The self-organizing loop depends on the monitoring process to take decisions. We defined a two-step monitoring process. On the first step all data passing through the measurement points is stored. Indeed, we don't store the data itself, but the size of the packets received/sent and the size of blocks of memory/storage read and written are saved in a first-stage buffers. Distinct buffers are always used for network-related and memory/storage-related resources during the monitoring process. The second step is activated periodically and for the experiments we use an interval of 10s. All size of the packets or blocks of the first-stage buffers are summed and stored in the second-stage buffers. At the end of the second-step, the first buffers are erase, and no history of the received/sent or read/write data is kept for the next second-step monitoring process.

When the self-organizing control loop starts a cycle, it first determines the amount of network and memory/storage resources consumed within two self-organizing loops. The amount of used resources is the average obtained after processing data inside the second-stage buffers. Differently from the monitoring process, at the end of a self-organizing cycle we don't clear the second-stage buffer. We use a sliding window to keep part of data inside the second-stage buffer and erase the rest of the data.

V. Evaluation

We defined 2 sets of experiments to be executed with the testbed. The first experiment presents the behavior of

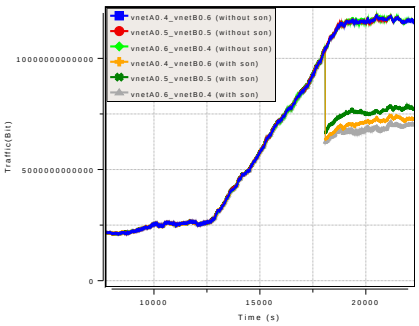

Fig. 6. Total traffic of the substrate network.

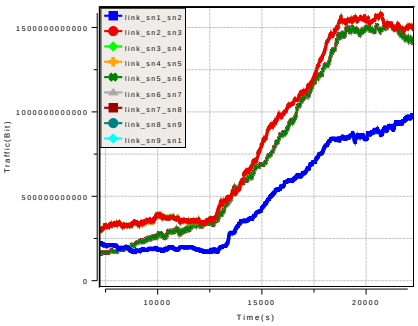

Fig. 7. Traffic per substrate link. (Without son - 0.4_VN#1_0.6_VN#2)

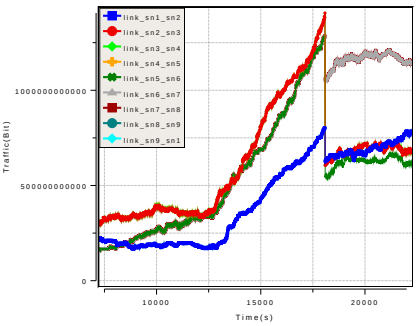

Fig. 8. Traffic per substrate link. (With son - 0.4_VN#1_0.6_VN#2)

self-organizing model when the request rate change from a lower request rate to a higher request rate. The second set of experiments verifies the efficiency of the self-organizing model when the interval to activate the self-organizing control loop varies from 1s to 20s, and when the preferences of the movies are different for each virtual network. Table II presents the parameters of the substrate and the virtual networks that are fixed for both sets of experiments.

Parameter	Value
Total number of substrate nodes	9
Number of virtual networks	2
Number of virtual nodes per VN	3
Data rate of substrate links	2Gbit/s
Data rate of virtual link	1Gbit/s
Size of virtual storage	50GB
Size of each movie	4GB
Overloaded link threshold α	0.7
Similarity factor δ	0.2
Type of IPTV transmission	Video on demand
Number of communities	2
Number of movies per community	3
Duration of movie	100 min
Placement of the movies for each virtual network	Movie#1 $\vdash$ Virtual node A Movie#2 $\vdash$ Virtual node B Movie#3 $\vdash$ Virtual node C
Preference I (P-I)	Movie#1 - 30% Movie#2 - 20% Movie#3 - 50%
Preference II (P-II)	Movie#1 - 50% Movie#2 - 20% Movie#3 - 30%
Preference III (P-III)	Movie#1 - 30% Movie#2 - 50% Movie#3 - 20%

TABLE II
FIXED PARAMETERS

A. Change on Request Rate

At the beginning of the experiment the total request rate λ of the testbed is fixed to 100 req/hour. This amount of requests is shared by the two virtual networks. We executed the experiments with three groups of shared λ per virtual network: in group 1 VN#1 has 40% and VN#2 has 60% (0.4_VN#1_0.6_VN#2); for group 2 each virtual network has 50% of λ (0.5_VN#1_0.5_VN#2); and group 3 has 60% for VN#1 and 40% for VN#2 (0.6_VN#1_0.4_VN#2). After 3 hours of transmission we changed the total λ to 500 req/hour, but we keep the percentage of the shared λ after the change. In this experiment, the interval among the self-organizing cycles is fixed to 10s, and the preference of the movies is fixed as P-III for both virtual networks. Figs.6-8 illustrate the associated results of this experiment.

Fig.6 brings the total traffic of the substrate network with and without the self-organizing model. The three groups of λ present the same behavior when the self-organizing model is not active. In this case the total traffic consumed in the substrate network goes from [1.8; 2.7] Gbit/s up to [11.6; 12.1] Gbit/s, during a period of 100 min that represents the interval to transfer a entire movie. When the self-organizing model is active in the network we can see on Fig.6, that before the end of the transmission of one movie the reorganization of the virtual resources occurs. After the re-organization, the total traffic consumption decreases approximately 4.2Gbit/s.

Fig.7 and Fig.8 describe the traffic consumption per substrate link when the distribution of λ is given by group 1 (0.4_VN#1_0.6_VN#2). When the self-organizing in not active (Fig.7) there are three main flows being transmitted. The first one is passing through “link_sn1_sn2”, “link_sn8_sn9”, and “link_sn9_sn1”. These links contain the requests for movies 1 and 3 from virtual nodes A#1,A#2 to C#1,C#2. The second flow is related to requests for movie 2 (hosted on virtual nodes B#1 and B#2) from the users on virtual nodes A#1 and A#2. This traffic is passing through “link_sn2_sn3”, “link_sn3_sn4”, and “link_sn4_sn5”. Finally the third flow is associated with the requests for the movie 3 (hosted on virtual nodes C#1 and C#2) from the users on virtual nodes B#1 and B#2, and the traffic is flowing through “link_sn5_sn6”, “link_sn6_sn7”, and “link_sn7_sn8”.

In this scenario of test, the self-organizing model detects that substrate links around substrate node sn_5 are overloaded. The preference P-III makes the movies hosted in this substrate node more popular, and thus there is more traffic going out from this substrate node. Moreover, P-III also determines that the second more requested movie is hosted inside substrate node sn_2. So, users from substrate node sn_5 tends to request more movies in sn_2 than in sn_8 (Fig.5) for both virtual networks, and this makes the traffic in the left size of sn_5 be higher than on the right side (Fig.5). The next step is to identify the

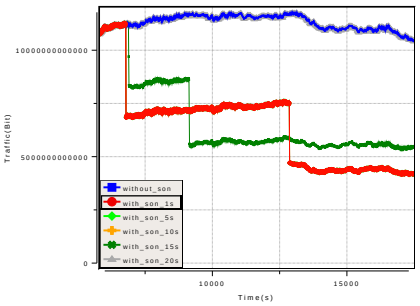

Fig. 9. Scenario VN#1_P-I_VN#2_P-I

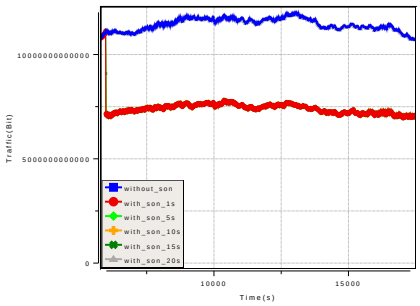

Fig. 10. Scenario VN#1_P-I_VN#2_P-II

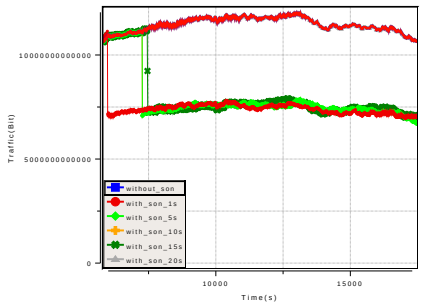

Fig. 11. Scenario VN#1_P-I_VN#2_P-III

virtual node that is consuming more resources. The graphic presented in Fig.8 has VN#2 with 60% of the total request rate, and thus this virtual network is consuming more resources of the substrate network, and is identified by the self-organizing model to be moved.

Finally, we need to know which virtual nodes of VN#2 should be moved. The self-organizing algorithm is executed and at the end of the cycle two virtual nodes of VN#2 were migrated. The number of virtual nodes that move is transparent for the system. Analyzing the log files it is possible to identify which virtual nodes moved to which substrate nodes. In this scenario of test, virtual node A#2 moved from sn_2 to sn_3, and virtual node B#2 moved from sn_5 to sn_4.

The self-organizing model reacted in the expected way for traffic load imposed by the request model configured for this experiment. We concluded that differences between the request load of distinct virtual networks (like 20% of the total request load) does not present a significant impact on the total traffic consumption. We also concluded that the preferences of the movies play an important role on the decision of which virtual node should be moved, and where to move this virtual node.

B. Varying Preferences and Execution Interval of Self-organizing Control Loop

For this experiment we fixed the total number of requests in the substrate network to 500 req/hour, and each virtual network has 250 req/hour. There are two dimensions varying. The first one is the interval of self-organizing execution that varies among 1s, 5s, 10s, 15s, 20s. The second dimension is related to the preferences of the movies, and we present the combination of the preferences between the virtual networks. Figs.9-11 show the total traffic consumed in the network for this set of experiment.

Comparing the graphics of Figs.9-11 it is visible that the preferences of the movies impact the self-organization efficiency. Fig.9 presents the case where the users of both virtual networks have the same preference for the movies. The consequence is that the scenario in Fig.9 is more overloaded then the others. The other two scenarios, respectively from Fig.10 and Fig.11, take just one round to reorganize the virtual nodes. However, the graphic of Fig.9 shows that two rounds of reorganization of the virtual nodes are required, and the resultant traffic is lower then the other cases. For example, after the last reorganization the traffic consumption in "VN#1_P-I_VN#2_P-II" and "VN#1_P-I_VN#2_P-III" is within the interval of [7; 7.7]Gbit/s In contrast, "VN#1_P-I_VN#2_P-I" takes more time to reach the stability, but the resultant traffic in the substrate network is within [4.3; 4.8]Gbit/s. These experiments indicate that the more overloaded is the substrate network the better will be the efficiency of the self-organizing model.

Now, if we analyze the graphics of Figs.9-11 considering the interval of self-organizing execution, we also verify that it plays an important role in our model. In all scenarios there are re-organizations when the interval of self-organizing execution is smaller than 20s. This behavior indicates that there is a relationship between the monitoring process and the self-organizing model. The monitoring process used for this experiments is based on sliding windows that keep part of the measurements executed in past cycles of self-organizing control loop, and can perhaps masquerade or postpone required self-organizations. An future study about the relationship between monitoring process and self-organizing might improve the efficiency of the self-organizing model.

An interesting behavior is observed on scenario "VN#1_P-I_VN#2_P-I" (Fig.9). The execution of this scenario with interval of self-organization of 1s, 5s, and 10s presents the same curves (from now on we will refer just the interval of 10s represented by the case with_son_10s). The same scenario with interval of 15s (with_son_15s) presents distinct results after the self-organization. The cases with_son_10s and with_son_15s execute the first round of reorganization approximately in the same time (with_son_10s executes the reorganization 1.5 min before with_son_15s). Both cases execute two rounds of self-organization but the second one occurs in different moments.

The case with_son_15s executes the second round of self-organization approximately 38 min after the first round, and reaches a stable state where the overall traffic consumption on the network is within [5.3;5.8]Gbit/s. Case with_son_10s takes 101 min to execute the second self-organization round, and then it also reaches a stable state with traffic consumption within [4; 4.5]Gbit/s. The maximum reduction on traffic consumption achieved by with_son_15s is 52%, while with_son_10s reaches 62% (if compared with the traffic consumption when the self-organizing is not active). The early execution self-organization on case with_son_15s does not guaranteed that it would present a better efficiency overtime. As evidenced by case with_son_10s, a later reorganization reduced even more the total traffic consumed in the substrate

network. To understand the reason for this difference we have to analyze the behavior of the each substrate link. Fig.12 presents the traffic inside the substrate links when case with_son_10s is executed, and Fig.13 illustrate the traffic for case with_son_15s.

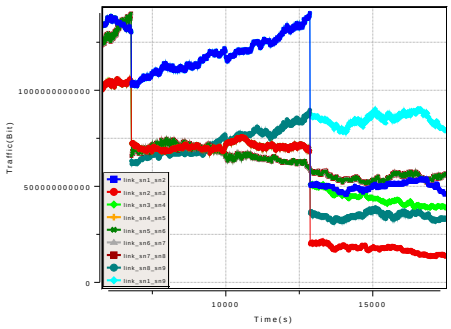

Fig. 12. Scenario VN#1_P-I_VN#2_P-I - Case with_son_10s

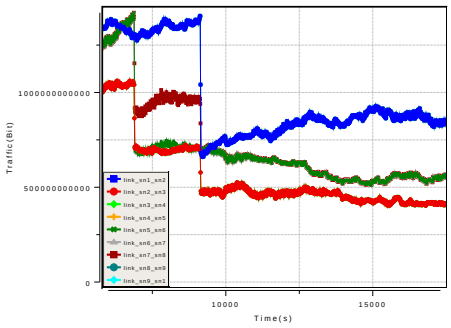

Fig. 13. Scenario VN#1_P-I_VN#2_P-I - Case with_son_15s

The main differences between Fig.12 and Fig.13 are the substrate links affected by the self-organizing cycle. The initial round of self-organization on both cases is triggered by "link_sn7_sn8", however the movement of virtual nodes was not the same. Case with_son_10s moved 2 virtual nodes on the first round, while case with_son_15s moved only 1 virtual node. Decisions of which virtual nodes should be moved in a round of self-organization cause consequences on all subsequent decisions. In this sense, the the interval of executing the self-organization control loop is a relevant parameter when there is an overloaded scenario. This conclusion is supported by the fact that cases with_son_10s and with_son_15s on the other two scenarios (Fig.10 and Fig.11) present a more stable behavior. This experiment shows that an specific study about the execution interval of self-organizing control loop is required to determine the parameters that produce the best results considering different loads on the substrate network.

VI. Related Work

On a first sight, the proposed self-organizing model might be seen an extension of existing virtual machine live migration. Recently, self-organization techniques [16][17] have been employed on server virtualization scenarios [18][3][19]. In these cases, the virtual machines are self-organized according to the workloads of the physical nodes, and generally, this self-organization is accomplished migrating virtual machines to physical ones with lower workloads. However, the metrics traditionally used to determine the workload of virtual machines are CPU and memory, and in a virtual network the bandwidth consumption is one major metric to be considered in the migration process. Beyond these metrics, virtual network live migration is different from virtual machine migration because it has also to deal with virtual topology issues and routing connections reconfigurations.

A very recent research on virtual router migration is presented by Yi Wang et al. [6]. In this paper the authors proposed VROOM, a virtual router migration mechanism, where the virtual interfaces of the routers are not directly mapped to physical ports and in this sense it is possible to migrate a router among different physical devices. The authors presented the migration mechanism itself and argued the advantages of using this approach to deal with management changes, planning, and new service deployment. However, nothing was mentioned about the analysis to trigger the router migration, and how this approach can help to reduce the resource consumption on the substrate network.

Since network virtualization is a very new research area, there are few proposals properly considering maintenance of resource management. Until now the focus of the researches are the efficient mapping of virtual networks to the substrate network. The work presented by Houidi et al. [10] explicitly considers autonomic principals on their mapping process. On the other hand, Yuy et al. [9] do not use autonomic features, but the authors employ splitting and migration mechanism to reorganize the mapping of virtual links used by each virtual network.

In [10], Houidi et al. present a distributed and autonomic mapping framework responsible for self-organizing the virtual networks on top of the substrate network every time a new deployment request arrives. This request triggers the autonomic elements, which in their turn, exchange messages to build a global view of the all virtual network topologies and decide where to place/replace the resources of the virtual networks. Despite the fact that the approach presented in [10] employs autonomic features and decentralization on the reallocation mechanism itself, the decision of when and how reorganize the substrate network is still based on external and global view approach. As discussed before, this design is not appropriated to deal with dynamic, transient changes on resource consumption.

The work presented by Yuy et al. [9] is specially target to deal with dynamic requests for embedding/removing virtual networks from the substrate. The authors map the constrains of the virtual network to the substrate network by splitting the requirements of one virtual link in more than one substrate link. The reorganization is accomplished migrating/realocating split virtual links to other substrate network links. A time window is used to regulate when a reorganization of the virtual network links is required. The problem of this approach is defining an appropriate time window able also to deal with the dynamics of the virtual network life cycle, and not only with the dynamics of the requests to embed.

Splitting and migrating virtual links are actions comparable to rerouting techniques on previous virtual networks environments [20]. However, network virtualization enables the resource management beyond rerouting traffic. As presented in this paper we can migrate virtual nodes to better use the substrate resources. Actually, the authors of [9] mentioned that the migration of virtual nodes is their future work. We believe that the our self-organizing model and the mapping model defined by Yuy et al. are complementary solutions for the efficient resource management of network virtulization technology.

VII. Conclusion

In this paper we presented a self-organizing model for substrate networks based on network virtualization concepts. Our work contributes to the research community on autonomic communications with lessons of how to build a self-organizing network without global knowledge or synchronization among the autonomic entities. The experiments proved that even without external management entities, or global view of the substrate network, it is possible to efficiently manage the resources. Our model also contributes to the network virtualization field. We showed that efficient maintenance of the substrate network can be achieved through the reorganization of the virtual resources. As part of future work we plan to introduce new self-* features in our model. Indeed, the next step of this research is the definition of a self-configuration and a self-optimization process to adapt the parameters of the self-organizing control loop and improve the success of a reorganization. Other directions of our work are related to the network virtualization context. We intend to investigate the cross-layer interactions between the management tasks executed on the substrate network and within the virtual network.

Acknowledgement

This work was partly funded by the Brazilian Ministry of Education (MEC/CAPES, PDEE Program, process 4436075), and by the European Union through the 4WARD project in the 7th Framework Programme [12]. The views expressed in this paper are solely those of the authors and do not necessarily represent the views of their employers, the 4WARD project, or the Commission of the European Union.

References

[1] S. Dobson, S. Denazis, A. Fernández, D. Gaïti, E. Gelenbe, F. Massacci, P. Nixon, F. Saffre, N. Schmidt, and F. Zambonelli, "A survey of autonomic communications," *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, vol. 1, no. 2, pp. 223–259, December 2006.

[2] P. k. Mckinley, F. A. Samimi, J. K. Shapiro, and C. Tang, "Service Clouds: A Distributed Infrastructure ofr Constructing Autonomic Communication Services," in *DASC'06: Proceeding of the 2nd IEEE International Symposium on Dependable, Autonomic and Secure Computing*, 2006, pp. 341–348.

[3] M. Steinder, I. Whalley, D. Carrera, I. Gaweda, and D. Chess, "Server virtualization in autonomic management of heterogeneous workloads," in *Proceeding of 10th IFIP/IEEE International Symposium on Integrated Network Management (IM 2007)*, May 2007, pp. 139–148.

[4] T. Guan and E. Zaluska, "An Autonomic Service Discovery Mechanism to Support Pervasive Device Accessing Semantic Grid," in *Proceedings of The 4th IEEE International Conference on Autonomic Computing (ICAC2007)*, Jacksonville, Florida, USA, 2007.

[5] N. Niebert, I. E. Khayat, S. Baucke, R. Keller, R. Rembarz, and J. Sachs, "Network Virtualization: A Viable Path Towards the Future Internet," *Journal Wireless Personal Communications*, vol. 45, no. 4, pp. 511–520, June 2008.

[6] Y. Wang, E. Keller, B. Biskeborn, J. van der Merwe, and J. Rexford, "Virtual routers on the move: live router migration as a network-management primitive," in *SIGCOMM '08: Proceedings of the ACM SIGCOMM 2008 conference on Data communication*. New York, NY, USA: ACM, 2008, pp. 231–242.

[7] N. Feamster, L. Gao, and J. Rexford, "How to lease the internet in your spare time," in *ACM SIGCOMM Computer Communications Review*, Jan 2007.

[8] S. Han, S. Lisle, and G. Nehib, "IPTV Transport Architecture Alternatives and Economic Considerations," *IEEE Communications Magazine*, vol. 46, no. 2, pp. 70–77, 2008.

[9] M. Yu, Y. Yi, J. Rexford, and M. Chiang, "Rethinking virtual network embedding: substrate support for path splitting and migration," *SIGCOMM Computer Communications Review*, vol. 38, no. 2, pp. 17–29, 2008.

[10] I. Houidi, W. Louati, and D. Zeghlache, "A Distributed and Autonomic Virtual Network Mapping Framework," in *Proceedings of Fourth IEEE/IFIP International Conference on Autonomic and Autonomous Systems, 2008. ICAS 2008*, March 2008, pp. 241–247.

[11] GENI, "Global environment for network innovations," 2008, available at http://www.geni.net/office/office.html.

[12] 4WARD, "The fp7 4ward project," 2008, available at http://www.4ward-project.eu/.

[13] C. Marquezan, J. Nobre, L. Granville, G. Nunzi, D. Dudkowski, and M. Brunner, "Distributed reallocation scheme for virtual network resources," in *IEEE International Conference on Communications (ICC 2009)*, 2009, Submitted.

[14] N. Degrande, K. Laevens, D. D. Vleeschauwer, and R. Sharpe, "Increasing the User Perceived Quality for IPTV Services," *IEEE Communications Magazine*, vol. 46, no. 2, pp. 94–99, 2008.

[15] D. Agrawal, M. S. Beigi, C. Bisdikian, and N. Lee, "Planning and Managing the IPTV Service Deployment," in *Proceeding of 10th IFIP/IEEE International Symposium on Integrated Network Management (IM 2007)*, 2007, pp. 353 – 362.

[16] K. Hermann, "Self-organizing replica placement - a case study on emergence," in *IEEE First International Conference on Self-Adaptative and Self-Organizing Systems (SASO 2007)*, 2007.

[17] J.-L. Lu, F. Valois, and D. B. M. Dohler, "Fisco: A fully integrated scheme of self-configuration and self-organization for wsn," in *IEEE Wireless Communications and Networking Conference, 2007.WCNC 2007*. IEEE Computer Society, March 2007, pp. 3370 – 3375.

[18] A. Singh, M. Korupolu, and D. Mohapatra, "Server-storage virtualization: integration and load balancing in data centers," in *SC '08: Proceedings of the 2008 ACM/IEEE conference on Supercomputing*. Piscataway, NJ, USA: IEEE Press, 2008, pp. 1–12.

[19] X. Y. Wang, D. Lan, X. Fang, M. Ye, and Y. Chen, "A resource management framework for multi-tier service delivery in autonomic virtualized environments," in *IEEE/IFIP Network Operations and Management Symposium (NOMS 2008)*, 2008, CDROM.

[20] Y. Ohsita, T. Miyamura, S. Arakawa, S. Ata, E. Oki, K. Shiomoto, and M. Murata, "Gradually Reconfiguring Virtual Network Topologies Based on Estimated Traffic Matrices," in *Proceedings of 26th IEEE International Conference on Computer Communications. IEEE INFOCOM 2007*, May 2007, pp. 2511–2515.

Clarissa Marquezan Biography text here.

Giorgio Nunzi Biography text here.

Marcus Brunner Biography text here.

Lisandro Granville Biography text here.