Virtual Network Mapping into Heterogeneous Substrate Networks

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*—As the interest on Network Virtualization continues to grow, so does the awareness of the many technical obstacles to transpose before the envisioned virtualized network environment may become a reality. A significant obstacle lies on the efficient assignment of virtual resources into physical ones. Performing the so-called mapping of a virtual network into a substrate network is a computationally intensive task, due to the dual optimization required for nodes and links placement.**

The purpose of this paper is to tackle this problem taking into consideration real-life scenarios of network operators, where the limitations imposed by the heterogeneity of the virtual and substrate networks must be accounted for. To that end, this paper proposes a heuristic algorithm for virtual resources mapping in the physical infrastructure that supports the heterogeneity of networks, in both links and nodes. The mapping heuristic was evaluated both through simulation and in a real experimental virtualization platform. Through the simulation results, it is shown that the mapping approach is able to embed a high percentage of the Virtual Network (VNet) requests respecting all links and node constraints. With respect to the experimental results, the proposed algorithm was shown to be fast, requiring a mapping time in the order of low tens of milliseconds, and linearly scalable with the increase in the number of existing VNets.

***Index Terms*—Virtualization, Virtual Networks, Network, Embedding, Mapping, Heterogeneous**

I. Introduction & Motivation

Network Virtualization has been hailed by the research community as a key enabler technology to escape from the current well-known limitations of the Internet. Moreover, it is also seen as a viable tool for experimenting novel network protocols on production networks without affecting other critical services, running of the same substrate network. The capability of providing isolated and protocol-independent virtual networks on top of a single network infrastructure has also drawn attention from network operators. From one side they may reduce their Total Cost of Ownership (TCO) through consolidated network resources and management; from the other side they may gain business advantages by being able to provide custom-tailored networks to their costumers quickly and without significant infrastructure costs [1], [2].

Although there is a large interest on virtualized networks both from the research community and network operators, several challenges still prevent it from being deployed on real environments [3]. One of the major obstacles lies in the efficient embedding of a virtual network onto a substrate network. Since this process requires the simultaneous optimization of virtual nodes and links placement, it is complex in nature and requires large amounts of computing power. Some authors, such as [4], [5], [6], [7], [8], have already proposed possible solutions to this problem, but have failed to provide an algorithm that takes into consideration the limitations imposed by the heterogeneity of resources in terms of links (their bandwidth, delays, etc.) and nodes (CPU, RAM, etc.).

In this paper we propose a mapping algorithm, heuristic based, that considers the support of heterogeneous virtual and substrate networks. This algorithm defines stress formulas, for both nodes and links, that take into account the several characteristics of both nodes and links. The mapping approach is then performed heuristically using a Constrained Shortest Path First (CSPF) algorithm, which will then determine a node potential to be used in the mapping process. The mapping algorithm was evaluated both through simulation and through a real virtualization platform. From the simulation results, we show that the mapping approach is able to embed a high percentage of the VNet requests, while respecting all links' and nodes' constraints. The experimental results show that, unlike most available algorithms that are very time consuming and cannot be deployed in real scenarios, our algorithm is indeed feasible to be used in a production environment.

The paper starts with the discussion of the related work on mapping algorithms, on Section II, and proceeds on Section III with the specification the developed algorithm. Section IV analyzes its performance, both through simulation and through a real testbed environment. The paper concludes on section V.

II. Related Work on Mapping Algorithms

When receiving VNet request, it is of the Infrastructure Provider (InP) (who owns and manages the physical network infrastructure) best interest to optimize resource allocation in order to reduce congestion and maximize profitability (by allowing more VNets request to be accepted on the future). Efficient resource mapping must therefore deal with the simultaneous optimization of the placement of virtual nodes and links on a substrate network.

This simultaneous optimization can be formulated as an unsplittable flow problem [4], [9], known to be *NP-hard* , and therefore is only tractable for a small amount of nodes and links. In order to solve this problem, several approaches have been suggested, mostly considering the *offline* version of the problem where the VNet requests are fully known in advance.

In [5] a backtracking method based on subgraph isomorphism was proposed; it considers the online version of the mapping problem, where the VNet requests are not known in advance, and proposes a single stage approach where nodes and links are mapped simultaneously, taking constraints into consideration at each step of the mapping. When a bad mapping decision is detected, a backtrack to the previous valid mapping decision is made, avoiding a costly re-map.

Some works, such as [6], defined a set of premises about the virtual topology, i.e. the backbone nodes are star-connected and the access-nodes connect to a single backbone node. Based on these premises, an iterative algorithm is run, with different steps for core and access mapping. However, the algorithm can only work for sepecific topologies.

A distributed algorithm was studied in [10]. It considers that the virtual topologies can be decomposed in hub-and-spoke clusters and mapped each cluster independently, therefore reducing the complexity of the full virtual network mapping. But it still lacks from performance and scalability, when comparing with centralized approaches.

Zhu et al. [4] proposed a heuristic, centralized, algorithm for dealing with VNet embedding. The goal of the algorithm is to maintain a low and balanced stress of both nodes and links of the substrate network. However, the stress of nodes and links do not consider heterogeneity on their characteristics.

Yu et al. [7] propose an embedding algorithm which considers finite resources on the physical network, and enables path splitting (i.e. virtual link composed by different paths) and link migration (i.e. to change the underlying mapping) during the embedding process. Although, this level of freedom can lead to a level of fragmentation that is infeasible to manage on large scale networks. In [8], it was taken a formal approach to solve the online VNet embedding problem using a mixed integer programming formulation in a two-step approach. This approach, besides not supporting heterogeneity of nodes, is still significantly complex in real environments.

Although all these algorithms provide a solution for the virtual network mapping problem, most of them fail to take into consideration that not all virtual nodes are the same, they can have different requirements for Central Processing Unit (CPU), memory and location, and their links may be constrained by several parameters. The heterogeneity of both virtual and substrate resources is mostly not considered. Moreover, to the best of our knowledge, the proposed approaches were not tested in a real environment in a virtualization platform.

III. Mapping Algorithm

The proposed algorithm is inspired on the concepts of nodes and links stress defined in [4]. However, both the stress formulas and the algorithm itself are distinct from the ones presented in [4] for the following reasons: (1) the node stress is simply considered to be the number of running virtual machines and fails to consider the reality of physical nodes, where the CPU load, core count, frequency, and available RAM amount are important factors; (2) the link stress is determined considering the number of virtual links going through a physical link, instead of taking into consideration the reserved link's bandwidth; (3) it only takes into account perfectly homogeneous physical and virtual networks, as is reflected on the single pool of candidates for virtual resources; (4) node constraints, such as location and required specifications, are not contemplated and neither are the limitations associated with links' bandwidths, which are finite.

Our proposed algorithm is defined taking into account the previous issues. The algorithm starts by determining a link and node stress factor. Links and nodes with less stress are more prone to accepting new virtual resources. Notice that these concepts are similar in [4]; however, our stress formulas are different, taking into account the heterogeneity issues.

A pseudo-code description of the proposed algorithm is shown in algorithm 1.

Defining $k_j = 0...(L_{V_j} - 1)$ and $i = 0...(L_S - 1)$ where k_j is the link number of a given virtual link belonging to the j^{th} VNet, L_{V_j} is the number of virtual links in the same VNet, i is the link number of a given physical link and L_S is the number of links of the Substrate Network, one can start by establishing that the virtual link stress (S_{LV_j}) of the link k_j belonging to the j^{th} VNet is equal to its allocated bandwidth : $S_{LV_j}(k_j) = BW(k_j)$. This procedure is demonstrated on the pseudo-code on lines 1 to 9.

After all virtual links stresses are determined, the physical link stresses are calculated: $S_{LS}(i)$ is the link stress of the i^{th} physical link and is defined as (lines 10 to 12):

$$S_{LS}(i) = \sum_{j}^{N_V} \sum_{k}^{L_{V_j}} ((S_{LV_j}(k_j) | k_j \supseteq i)) \tag{1}$$

where N_V is the number of existing VNets. As opposed to the proposed link stress formula in [4], this formula takes into consideration the different allocated link bandwidths, and uses them to weight each physical link's stress.

Afterwards, it proceeds with the determination of Node Stress (S_N - lines 13 to 15), which is a combination of the currently available Substrate Node resources and weights active Virtual Machines, free RAM (Free RAM) amount in MB, number of CPUs (N.CPU), CPU frequency in MHz (CPU Freq.) and current CPU Load, which varies between 0 and *N.CPU* (number of physical CPU Cores).

The Λ function, defined in 2, determines whether or not a virtual node n_j, belonging to the j^{th} VNet, is active and running on the i^{th} physical node.

$$\Lambda(n_j, i) = \begin{cases} 1 & \text{if } (n_j \supseteq i) \wedge (n_j \text{ is active}) \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

The Node Stress of the i^{th} physical node is:

$$S_{N_i} = \frac{\sum_{j}^{N_V} \sum_{n}^{N_{V_j}} \Lambda(n_j, i)}{\delta + \text{Free RAM} \cdot \text{CPU Freq} \cdot (\text{N.CPU - Load})} \tag{3}$$

i.e., the total number of virtual machines running on the i^{th} physical node, divided by a metric of available resources. δ is a small constant to avoid dividing by 0, and N_{V_j} is the number of virtual nodes on the j^{th} VNet. Just like in the case of the

link stress formula, adjustments were also made that aim to consider not only the number of running virtual machines on each physical node, but also the capabilities and free resources of each node. By taking into account both the available RAM amount and a measure of available computing resources, this node stress formula should provide a better perspective on the usage and capabilities of the physical nodes.

The next step is the determination of node candidates (lines 16 to 22). For each virtual node, a set of physical candidates is determined based on eliminative constraints: location, number of CPUs, CPU frequency, free RAM amount, and available HDD space. The location constraint allows for the establishment of both logical groups and a physical location, based on a center GPS coordinate and a radius; thus reflecting a coverage area. The *MeetsConstraints* function on line 18 returns *true* if a given physical node meets all of the previously specified constraints.

After determining the candidates for each virtual node, a sorting algorithm is run that orders the virtual nodes by their number of candidates so that virtual nodes with fewer candidates will be mapped first (line 23).

The algorithm continues with the node mapping and path selection, according to lines 25 to 35. For each possible candidate v a CSPF algorithm to all other candidates (u) of the virtual neighbour nodes is calculated using the previously calculated Link Stresses as weights, and the path cost is stored ($D(v,u)$ - line 30). The *node potential* $\pi(v)$ is then determined using the formula:

$$\pi(v) = \pi(v)\frac{\sum_{u \in V_C} D(v,u) \cdot S_{N_v}}{Count(n.Candidates)} \quad (4)$$

where V_C is a set containing the candidates of the neighbour virtual nodes. This equation produces a node potential that, at the end of each main loop (line 25), reflects a node potential averaged by the number of possible destination nodes, which assures that physical nodes with different amount of possible virtual links are treated fairly. In addition, this formula also differs from the one established in [4] due to its numerical behaviour: while the one previously proposed shows an hyperbolic behaviour with the increase in node stress, this one has a linear behaviour, and is, thus, more numerically balanced. Upon calculating the *node potential* of all candidates, the candidate with the lowest one is selected (line 35).

The algorithm terminates successfully when all the requested virtual nodes are properly mapped, each one on a different physical node, chosen from within its candidate set, and the best-constrained paths for each virtual link are determined.

Consider the simple situation of figure 1, where the stresses of physical nodes and links have already been calculated. If it is required to map a new virtual network, composed of virtual nodes *A* and *B*, with a single virtual link using a bandwidth of 20Mbps, the first step is to determine a set of physical candidates for each virtual node. Considering the constraints, *A* can be mapped either on *c* or *d*, and *B* can be mapped either on *a*, *b* or *c*.

Algorithm 1: Mapping Algorithm Pseudo-Code

```
   input : Substrate (Substrate Network) , V_Request (Requested VNet)
   output: V_Map (Mapped Virtual Network)
1  foreach Link i in Substrate.Links do
2      foreach VNet j in Substrate.VNets do
3          foreach Link k in j.Links do
4              if Link k_j ⊇ Link i then
5                  S_LS(i) += S_LV_j(k_j) ;
6              end
7          end
8      end
9  end
10 foreach Link i in Substrate.Links do
11     S_LS(i) = Σ_j^NV Σ_k^LV_j ((S_LV_j(k_j) | k_j ⊇ i)) ;
12 end
13 foreach Node i in Substrate.Nodes do
14     S_Ni = (Σ_j^NV Σ_n^NV_j Λ(n_j,i)) / (δ+Free RAM·CPU Freq·(N.CPU - Load)) ;
15     π(v) = 0 ;
16 end
17 foreach Node n in V_Request.Nodes do
18     foreach Node i in Substrate.Nodes do
19         if MeetsConstraints(n, i) then
20             n.Candidates.Add(i) ;
21         end
22     end
23 end
24 SortVirtualNodes(V_Request) ;
25 foreach Node n in V_Request.Nodes do
26     foreach Link k connected to n do
27         ConnectedVNode=GetLinkDestination(k) ;
28         foreach SourceCandidate v in n.Candidates do
29             foreach DestCandidate u in ConnectedVNode.Candidates do
30                 D(v,u)= Cost(CSFP_Dijkstra(v,u)) ;
31             end
32             π(v) = π(v) + (Σ_{u∈V_C} D(v,u)·S_Nv) / (Count(n.Candidates)) ;
33         end
34     end
35     n.Map = v : π(v) = min(π) ;
36 end
37 foreach Node n in V_Request.Nodes do
38     V_Map.Nodes ∪ n ;
39     foreach Link k connected to n do
40         ConnVNode=GetLinkDestination(k) ;
41         V_Map.Links ∪ CSFP_Dijkstra(n.Map,ConnVNode.Map) ;
42     end
43 end
```

Since *A* has the fewest number of physical candidates, the mapping procedure will start with this node. By applying the CSPF Dijkstra algorithm considering firstly *c* as a source node and *a* and *b* as potential destination nodes, the node potential of 0.7144 is determined. Applying the same procedure to the candidate *d* results in a node potential of 0.5417. Since *d* has the lowest node potential, it is chosen to hold the virtual node *A*. Regarding the virtual node *B*, mapped next, the same algorithm is applied, resulting with *a* being chosen to hold *B*. Finally, the CSPF Dijkstra algorithm decides that the link between *a* and *b* is the best to hold the requested virtual link.

IV. Performance Results

This section contains two types of performance results: simulation and experimental based. The simulation results assess the efficiency of the algorithm in determining the best mapping for the VNets requested. The experimental results assess the feasibility of the algorithm in a real environment and the cost of its deployment. It is known that most available algorithms are very time consuming and cannot be deployed

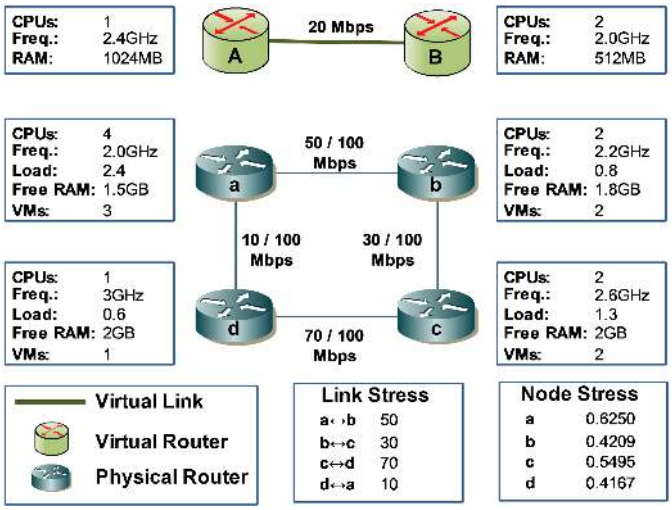

Figure 1. Virtual Network Mapping Example.

in real scenarios; in this experimental setup we show that it is feasible to use our algorithm in a production environment. Because no similar mapping algorithm was found in the literature, i.e. one that takes into account the heterogeneity of the substrate and virtual networks, no comparisons were performed with other algorithms.

A. Simulation Results

The purpose of this subsection is to simulate the developed algorithm in conditions as similar as possible to the ones found on real networks, with heterogeneous resources and links. By testing the algorithm on these "closer-to-reality" conditions, insight and conclusions about the algorithms performance and applicability shall be taken.

Regardless of the simulation parameters, the following simulations will all obey to a general procedure, which will be described next.

On the first step, a physical topology was generated using the Waxman random topology generation method [11], with 30 physical nodes. The recommended parameters, $\alpha = 0.4$ and $\beta = 0.4$, presented some connectivity issues, especially for reduced amounts of nodes, e.g. less than 16 nodes. These settings often caused isolated nodes or clusters where there was not at least one path between every physical node. Hence, after generating the topologies, in the lack of full connectivity, nodes with less links were given additional links until full connectivity was established.

The generated physical nodes were randomly attributed a set of parameters, from a pool of possible ones, such as RAM amount, number of CPUs and CPU frequency, using an uniform random distribution. The physical link's bandwidth was set at a fixed bitrate. Next, virtual networks were generated using the same model, with a varying amount of virtual nodes. After generating the virtual topology, the virtual nodes were also randomly attributed a set of specifications, with a uniform distribution. The virtual nodes' available specification pool can be observed on table I.

N. CPUs	{1; 2; 3; 4 }
CPU Frequency (GHz)	{2.0 to 3.2 in 0.1 steps }
RAM Memory (MB)	{64; 128; 256; 512 }
Link Bandwidth (Mbps)	{34.368 139.264 }

Table I
VIRTUAL NETWORK MAPPING- VIRTUAL NODES' PARAMETERS POOL.

In the virtual network mapping algorithm, if the mapping succeeds, the virtual nodes are placed on the physical nodes, reducing the amount of available RAM and increasing the physical nodes CPU load by a random amount. The utilization of the physical links will also increase according to the bandwidth utilized by the virtual links. Notice that we do not include a performance comparison with available mapping approaches, as they do not consider heterogeneous nodes and links.

In order to evaluate the mapping algorithm, two main metrics were considered: the node stress ratio R_N and the link stress ration R_L. From the definition of these two metrics on equations 5 and 6, one can see that in a perfectly load balanced network, their value should be 1. Therefore, smaller stress ratios indicate better virtual network embedding in the network. On the equations, N_S and L_S are the set of substrate nodes and links, respectively. A confidence interval of 95% was considered for every result.

$$R_N = \frac{\max_{v \in N_S} S_N(v)}{[\sum_{v \in V_S} S_N(v)]/\,|N_S|} \tag{5}$$

$$R_L = \frac{\max_{v \in V_S} S_L(v)}{[\sum_{v \in L_S} S_L(v)]/\,|L_S|} \tag{6}$$

1) Simulation Scenario 1: The first simulation scenario assumes that the parameters of the physical resources are taken from the pools of table II. For each generated physical network, attempts were made to map as many virtual networks as possible. It was considered that a mapping algorithm could not map any more virtual networks when it failed to embed 10 successive virtual networks.

N. CPUs	{2; 4; 6; 8}
CPU Frequency (GHz)	{2.0 to 3.2 in 0.2 steps }
RAM Memory (GB)	{2; 4; 6}
Link Bandwidth (Mbps)	{1000}

Table II
VIRTUAL NETWORK MAPPING SIMULATION SCENARIO 1- PHYSICAL NODES' PARAMETERS POOL.

Two approaches of the developed algorithm were simulated, one that starts the embedding by selecting the virtual nodes with the least amount of physical candidates, and another one that starts the mapping in a random fashion, i.e. without pre-sorting the generated virtual nodes.

The simulation was run 100 times for different virtual network sizes. The number of virtual nodes ranged from 4 to 14 in increments of 2.

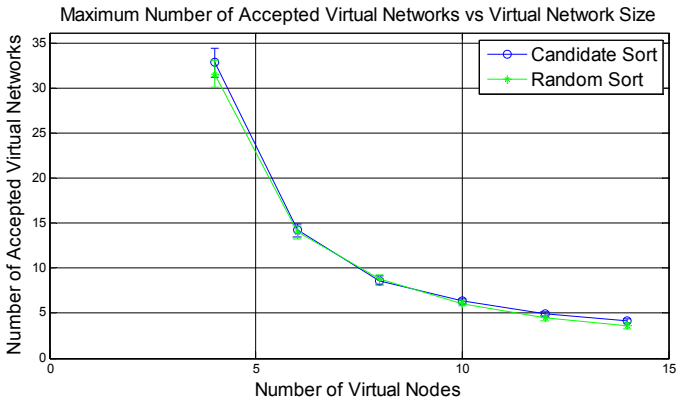

Figure 2. Virtual Network Mapping Simulation Scenario 1 - Maximum accepted Virtual Networks.

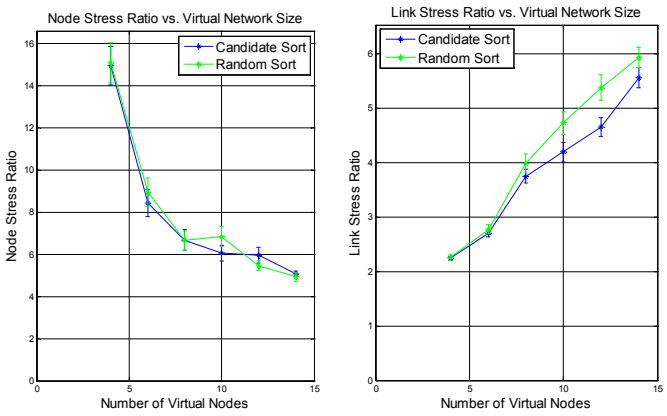

Figure 3. Virtual Network Mapping Simulation Scenario 2

N. CPUs	{4; 8; 12; 16}
CPU Frequency (GHz)	{2.0 to 3.2 in 0.2 steps }
RAM Memory (GB)	{4; 8; 12}
Link Bandwidth (Mbps)	{1000}

Table III
VIRTUAL NETWORK MAPPING SIMULATION SCENARIO 3- PHYSICAL NODES' PARAMETERS POOL WITH DOUBLED NODE CAPACITY.

The results may be observed on figure 2. As the virtual networks' size grows, due to the increase in the number of virtual nodes, the number of maximum accepted virtual networks decreases, which is as expected since larger networks are harder to map due to a higher amount of constraints. Considering the case of small virtual networks, with 4 virtual nodes for example, the mapping algorithms were able to embed about 33 of them, while in the case of virtual networks with twice the virtual nodes, it was only able to embed approximately 8.

The number of maximum accepted virtual network appears to behave similarly to a decaying exponential function, with the increase of the size of the virtual networks. The mapping algorithm performing a pre-sort of the virtual nodes, considering their amount of physical candidates, consistently shows slightly better results than the random one, particularly for simulations with few or many virtual nodes.

2) Simulation Scenario 2: Although the maximum number of accepted VNets is important, the load distribution should not be disregarded. To that end, this simulation scenario aims to evaluate the performance of both approaches (candidate and random sort) relating the node and link stress ratios, according to equations 5 and 6. In order to provide a fair comparison between the two approaches, an embed of 75% of the previously identified maximum of accepted virtual networks was performed.

The physical nodes' specifications were kept equal to the previous simulation ones. The simulation was run 300 times for different virtual network sizes. The number of virtual nodes ranged from 4 to 14 in increments of 2.

Starting with the figure 3, it is possible to state that, as the size of the virtual networks grows, the node stress ratio decreases, showing that the network's node load is better distributed. The large node stress ratio differences, regarding 4 and 14 nodes VNets, is mainly due to the way the node stress is calculated. Since the node stress is inversely proportional to the free CPU load, which varies between 0 and N_{CPUs}, one can realize that as the physical nodes become loaded, and their available CPU load tends to 0, the node stress will tend to infinity. In the 14-node virtual network embedding scenario, since only 3 virtual networks were embedded, for a total of 42 embedded virtual nodes, it was not very likely that a set of physical nodes got their available CPU load close to 0; thus, their node stress was kept at moderate levels.

On the other hand, since in the case of 4-node networks, 25 virtual networks with 100 virtual nodes were embedded, it was more likely that some physical nodes, possibly physical nodes with fewer CPUs, got overloaded and that their node stress reflected that overload, thus producing higher node stress ratios. The node stress ratios followed a similar trend, with the pre-sorting approach faring slightly better overall.

Regarding the evolution of the link stress ratio with the increase of the size of virtual networks, it is possible to note that it shows a growing behaviour. The reason for this behaviour is quite simple: when considering the embedding of smaller virtual networks, the granularity for link placement optimization is high; therefore, it will be easier to better take advantage of physical links with less link stress. In the case of larger and more complex virtual networks, there is less granularity in link placement, it is harder to optimize link placement due to node constraints.

Through the attained results, it is possible to state that pre-sorting the nodes leads to lower link stress ratios.

3) Simulation Scenario 3: In order to assess the impact of both the nodes' and links' capacity on the overall maximum of accepted virtual networks, the tests of the first simulation run were repeated considering two separate situations: the first one where the physical node's capacity was doubled, according to table III, and a second one where the physical links' capacity was doubled, as observed on table IV.

On the first case, figure 4, only a minor improvement in the number of accepted virtual networks was achieved with the

N. CPUs	{2; 4; 6; 8}
CPU Frequency (GHz)	{2.0 to 3.2 in 0.2 steps }
RAM Memory (GB)	{2; 4; 6}
Link Bandwidth (Mbps)	{2000}

Table IV
VIRTUAL NETWORK MAPPING SIMULATION SCENARIO 3- PHYSICAL NODES' PARAMETERS POOL WITH DOUBLED LINK CAPACITY.

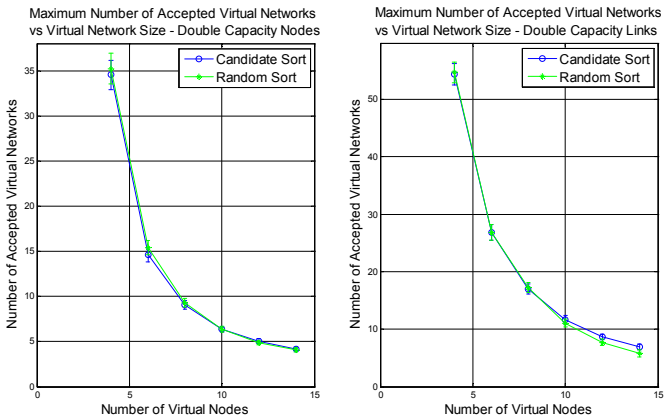

Figure 4. Virtual Network Mapping Simulation Scenario 3 - Maximum accepted Virtual Networks.

doubling of nodes' capacity. In the best-case scenario, for virtual networks composed of 4 virtual nodes, the improvement was limited to 2 additional embedded virtual networks. The differences in the mapping algorithm with or without candidate sorting are barely perceptible.

Regarding the doubling of link capacity, the gains realized are notorious. In fact, the number of accepted virtual networks almost doubled for virtual networks with 4 to 8 virtual nodes, and showed significant improvements for the other virtual networks' sizes.

Considering the accomplished results, it is clear that, for the specified simulation parameters, the main limiting factor for virtual network embedding is the links' capability. Improving the nodes' capacity barely showed any improvements, while increasing the links' bandwidth showed improvements similar to the bandwidth's increase factor.

B. Experimental Tests

In this section, the performance of the proposed algorithm will be evaluated in a real platform with the goal of assessing if it presents a viable option in production environments. For this purpose we developed a network virtualization testing platform, which will be described next.

1) Experimental Network Virtualization Platform: The Network Virtualization System Suite (NVSS) is a development and testing platform [12] that provides a framework for working with a virtualized network environment. This platform is highly modular in nature and provides the necessary functionalities to monitor, discover [13], deploy and manage virtual networks running on top of a substrate network. It is designed to run on Fedora Core 8 and Debian Lenny Linux distributions with the Xen kernel.

This virtualization platform is composed of three modules: the Agent module, the Manager module and the Control Centre module; their hierarchical decomposition can be analysed on figure 5.

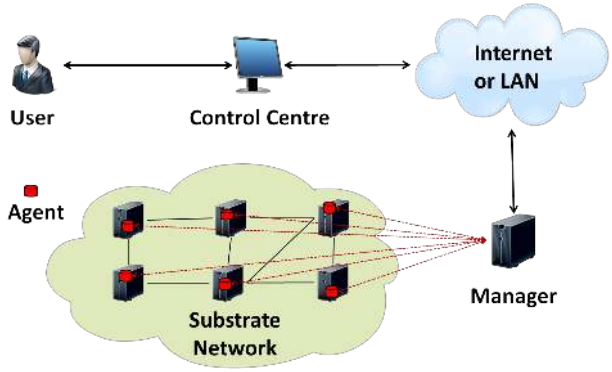

Figure 5. Network Virtualization System Suite - Existing modules

The Control Centre module is the user's front-end, i.e. the Graphical User Interface, from which the user may view and manage the existing VNets as well as design and create new ones.

The Manager module's functions are many-fold: it gathers information from the Agents and sends them commands, and keeps the Control Centre with up-to-date information about its requested virtual networks .

Finally, the Agent module is designed to run on every substrate node in order to act and periodically gather data from it. The Agents send their local resources' information to the Manager, provide discovery functions through a distributed algorithm, and execute resource creation and network configuration requests.

The testbed is composed of 6 physical nodes, containing Intel Core 2 Duo CPUs, and 6GB of RAM on average, which are interconnected using 100Mbps and 1Gbps links. 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.

2) Experimental Results: In order to assess the mapping times, 40 virtual networks were created, one at a time. For simplicity purposes, the created VNets were equal and their topology was an exact replica of the underlying physical network. Each virtual node was configured with 1 CPU, 64MB of RAM, and all virtual links were setup with a 1.0Mbps bandwidth. The Manager was running on an external computer, connected directly to the testbed (Intel P8600; 4GB of RAM; 100Mbps link).

The time required for the Manager to process the received request and return the mapping information was measured. Their average and 95% confidence interval were calculated .

The time required to perform the mapping is shown to increase with the number of existing virtual networks (figure 6). Since the mapping procedure only depends on the virtual

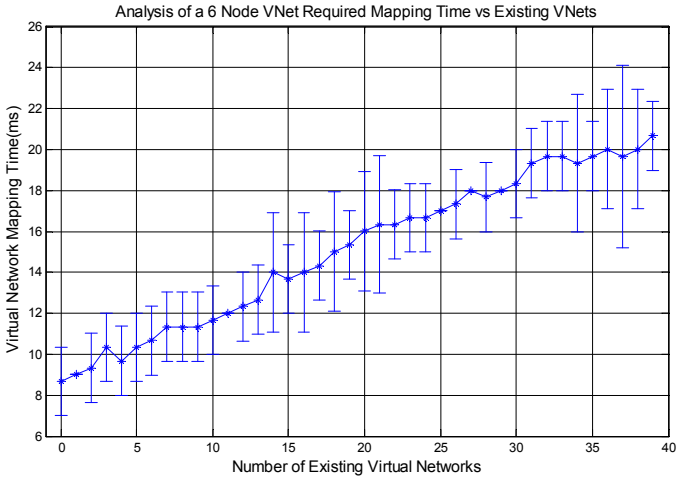

Figure 6. Virtual Network Mapping results.

network to be embedded and on the physical network, it would be expected that the mapping times remained constant, which is 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. Thus, for each additional virtual network, the Manager will need more time to calculate the physical links' stress. This increment in required time is revealed in the obtained results, which clearly show a linear scaling with the number of existing virtual networks.

Regarding the absolute mapping times, they remain 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. 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.

V. Conclusions

This paper proposed a heuristic-based virtual network embedding algorithm that considers the support of heterogeneous virtual and substrate networks. This algorithm defines stress formulas, for both nodes and links, that take into account the several characteristics of both nodes and links. The mapping approach uses a CSPF algorithm, which will then determine a node potential to be chosen in the mapping process.

The simulation results have shown that the main bottleneck for virtual network embedding is the capacity of the links. It was also shown that the number of embedded virtual networks has a decaying exponential behaviour with the growth of the virtual networks' size.The experimental results demonstrated that the proposed algorithm is fast (requiring a mapping time in the order of low tens of milliseconds) and linearly scalable with the increase in the number of existing virtual networks.

Future work will address the reconfiguration and fault-management approaches.

VI. 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] J. Carapinha and J. Jiménez, "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. 73–80.

[2] M. Melo, S. Sargento, and J. Carapinha, "Network Virtualisation from an Operator Perspective," *Proc Conf. sobre Redes de Computadores - CRC*, October, 2009.

[3] N. M. K. Chowdhury and R. Boutaba, "Network virtualization: State of the art and research challenges," *IEEE Communications Magazine*, no. 7, pp. 20–26, July.

[4] Y. Zhu and M. Ammar, "Algorithms for assigning substrate network resources to virtual network components," in *INFOCOM 2006. 25th IEEE International Conference on Computer Communications. Proceedings*, 2006, pp. 1–12.

[5] J. Lischka and H. Karl, "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*. New York, NY, USA: ACM, 2009, pp. 81–88.

[6] J. Lu and J. Turner, "Efficient mapping of virtual networks onto a shared substrate," Washington University in St. Louis, Tech. Rep., 2006.

[7] M. Yu, Y. Yi, J. Rexford, and M. Chiang, "Rethinking virtual network embedding: Substrate support for path splitting and migration," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, p. 17–29, 2008.

[8] N. Chowdhury, M. Rahman, and R. Boutaba, "Virtual network embedding with coordinated node and link mapping," in *INFOCOM 2009, IEEE*, 2009, pp. 783–791.

[9] D. G. Andersen, "Theoretical approaches to node assignment," Dec. 2002, unpublished Manuscript.

[10] I. Houidi, W. Louati, and D. Zeghlache, "A distributed virtual network mapping algorithm," in *Communications, 2008. ICC '08. IEEE International Conference on*, 19-23 2008, pp. 5634 –5640.

[11] B. Waxman, "Routing of multipoint connections," *Selected Areas in Communications, IEEE Journal on*, vol. 6, no. 9, pp. 1617 –1622, dec 1988.

[12] J. Nogueira, M. Melo, J. Carapinha, and S. Sargento, "Network Virtualization System Suite: Experimental Network Virtualization Platform," *TridentCom 2011, 7th International ICST Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities*, April 2011.

[13] J. Nogueira, M. Melo, J. Carapinha, and S. Sargento, "A Distributed Approach for Virtual Network Discovery," *IEEE Globecom 2010 Workshop on Network of the Future*, December 2010.