Noname manuscript No.
(will be inserted by the editor)

Optimal Mapping of Virtual Networks with Hidden Hops

Juan Felipe Botero · Xavier Hesselbach · Andreas Fischer · Hermann de Meer

Received: date / Accepted: date

Abstract Network Virtualization has emerged as a solution for the Internet inability to address the required challenges caused by the lack of coordination among Internet service providers for the deployment of new services. The allocation of resources is one of the main problems in network virtualization, mainly in the mapping of virtual nodes and links to specific substrate nodes and paths, also known as the virtual network embedding problem. This paper proposes an algorithm based on optimization theory, to map the virtual links and nodes requiring a specific demand, looking for the maximization of the spare bandwidth and spare CPU in the substrate network, taking into account the bandwidth demanded by the *hidden hops* when a virtual link is mapped. The components of the virtual networks (nodes and links) that do not ask for an specific demand are then allocated following a fairness criteria.

Keywords Network Virtualization · Virtual Network Embedding · Virtual Network Mapping · Optimization Theory

Juan Felipe Botero
Jordi Girona Street, 1 and 3, Barcelona, Spain
Tel.: +34-93-4017041
E-mail: jfbotero@entel.upc.edu

Xavier Hesselbach
Jordi Girona Street, 1 and 3, Barcelona, Spain
Tel.: +34-93-40159871
E-mail: xavierh@entel.upc.edu

Andreas Fischer
Innstr. 43, 94032 Passau, Germany
Tel: +49-851-509-3057
E-mail: andreas.fischer@uni-passau.de

Hermann de Meer
Innstr. 43, 94032 Passau, Germany
Tel: +49-851-509-3050
E-mail: demeer@fim.uni-passau.de

1 Introduction

The deployment of new Internet services is nowadays being more and more difficult, the lack of cooperation among stakeholders does not allow radical changes to the Internet architecture [3, 12, 11, 14]. This tendency is called ossification.

Network virtualization has been proposed as the alternative to face up ossification [12]. It allows multiple heterogeneous networks to cohabit on a shared physical substrate (SN)[1].

A Virtual Network (VN) - sometimes also called "Overlay Network" - consists of active and passive network elements realized on top of a substrate network. The active elements are called virtual nodes whereas the passive elements are called virtual links. Virtual nodes are interconnected through virtual links, forming a network that can be represented by a graph, where the virtual nodes are represented by the nodes in the graph and the virtual links are represented by the edges.

An instance of such a VN then is realized through a mapping of its elements to the substrate network. This mapping defines the relationship of virtual network elements to their respective counterparts in the substrate network. Several virtual nodes in the VN can correspond to a single node in the SN (i.e. the mapping is $n : 1$). Likewise, several virtual links (VL) can correspond to a single link in the substrate network. This $n : 1$ mapping is accomplished through resource sharing, with the resource being CPU time and memory for nodes and bandwidth for links.

This concept of virtual networks is realized in several different implementations. Examples are Virtual

[1] This paper will use indifferently the terms *substrate network* and *physical network*

Private Networks, Peer-to-Peer networks and networks virtualized with System Virtualization.

In Virtual Private Networks (VPNs), a dedicated (usually encrypted) Virtual Link is set up between two routers, possibly spanning several physical links. VPNs can be set up on different layers in the network stack, with the IPSec implementation on the network layer being one of the more popular ones [8].

In Peer-to-peer (P2P) overlay networks [13] an entire network structure is created on top of an existing network. P2P nodes correspond to Virtual Routers, setting up Virtual Links between them, that do not reflect the underlying network infrastructure, but rather the logical grouping of nodes within the P2P network.

In networks virtualized with System Virtualization [2], core routers host several operating systems with routing functionality. The routers are interconnected with Virtual Links that may again span several physical links. This approach allows for easy management and even mobility of Virtual Routers. Moreover, it becomes possible to deploy different network protocols alongside each other with the concept of system virtualization providing a clear compartmentalization. Besides, network virtualization provides reusable topology and an energy efficient scheme. Fig. 1 is an example of this implementation.

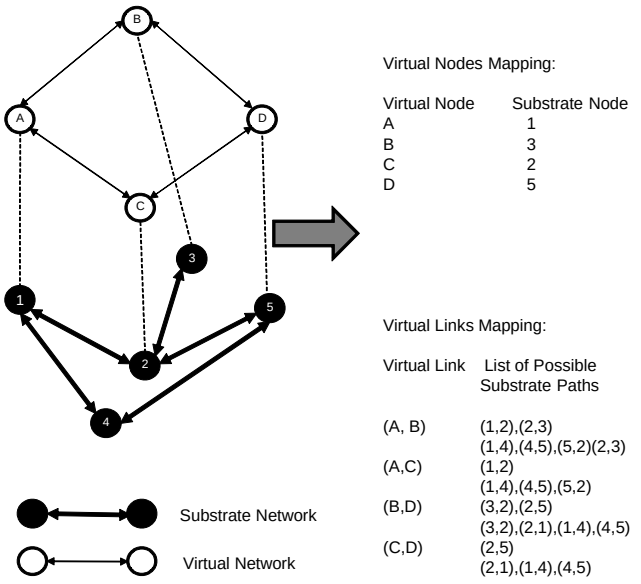

Fig. 1 Example of a Virtual Network Mapped Over a Substrate Network

The network embedding problem consists on the efficient mapping of a set of VN requests to substrate nodes and links. We define a VN request as a set of virtual nodes (with or without CPU requirements) that must be mapped to a set of SN nodes (i.e. nodes from the SN) with enough CPU resource to accomplish the requirements, and a set of virtual links (with or without bandwidth requirements) to be mapped to a set of SN paths (i.e. paths from the SN) accomplishing the required bandwidth demands and the *hidden hops* (intermediate substrate nodes) CPU demands. The network embedding problem is computationally hard to solve, as it will be shown in section 3, so the developed algorithm should follow a heuristic to try to obtain a near optimum value.

This problem turns out to be a form of the well known Multicommodity-Flow problem (MFP). This paper proposes an algorithm to solve the virtual network embedding problem based in optimization theory. The aim is to maximize the remaining SN resources (CPU and Bandwidth) while mapping the virtual nodes and links with specific demands. Apart from these demands constraints, the virtual nodes and links might impose more constraints, e.g. specific location in virtual nodes and delay constraints in virtual links. When the virtual nodes and links with specific demands are assigned, the algorithm assigns, in an equal way, the remaining substrate network resources to the virtual nodes and links without specific demand requests.

The paper is organized as follows: Next section describes the optimization model proposed to efficiently solve the embedding VN problem accomplishing the constraints and looking for the maximization of the spare bandwidth and spare CPU. Section 3 shows the complexity of the raised problem. Section 4 presents the heuristics used to solve the problem. Section 5 presents an example of the algorithm and finally, conclusions and future work are presented in section 6.

2 Optimization Model

2.1 Hypothesis

To accommodate a demand between two virtual nodes inside a virtual network, up to now, only one path is taken into account. This is often a realistic restriction due to the used routing protocol, or simply an explicit management requirement. However, multi-path approaches have been proposed. In [16] the virtual link demand is split among the possible paths, reducing, in this way, the computational complexity of the problem. Although this approach is computationally better, the difficulty of its implementation is higher [7].

In our model, the relationship among the elements of SN and VNs is as follows: One virtual router can be mapped to only one physical router, while a virtual link is represented by a path (group of consecutive links) in the SN. We consider that a physical node is a *hidden*

hop if it is part of a SN path mapped to a virtual link (i.e. if it is an intermediate node in the SN path).

Besides the CPU demand of some virtual nodes, we consider that each substrate node's, acting as a hidden hop (intermediate node) in a substrate path, that represents a virtual link, will have a CPU expenditure because it has to be configured and it will have to correctly forward the packets passing through this virtual link. The CPU resource that must be assigned to an intermediate node is a function of the virtual link demand. This resource expense had not been considered in previous studies [16,17,10].

This model assumes that the virtual nodes are assigned (mapped) to specific substrate nodes before the optimization process starts. This hypothesis is considered because we think that, for each virtual node, node mapping might not be done taking into account all the substrate nodes as potential candidates. At least, a virtual node must be subject to location constraints, so the potential candidates would be the nodes accomplishing these constraints. This aspect has not been taken into account in [10,17]. In future works, location constraints will be studied to map the most suitable substrate node, among a set of candidates, to a specific virtual node. The first attempt to include location constraints in node mapping can be found in [4].

The optimization model assumes virtual network request where there can be bandwidth and CPU fixed demands for some virtual links and nodes, but also the possibility that some other nodes and links do not ask for any resources is contemplated. After the demanded resources (bandwidth and CPU) in virtual links and nodes are assigned; a fair allocation is made, with the remaining resources, among the virtual links and nodes with no resources demanded. This fair allocation lies in allocates the shortest paths for the virtual links.

The required demand of the *hidden hops* and the mapping of undemanded request had not been taken into account in previous work [16,17,10,4]. These two improvements helps to carry out a more realistic virtual network mapping, because more virtual networks (even those without explicit demand requests) can be mapped taking into account the CPU demand of hidden hops.

2.2 Variables Definition

SN is represented by an undirected graph $G(V, E)$, where V is a set of elements called vertices and E is a set of elements called arcs or edges, with one or more numbers associated with each arc. Considering a ordered set of vertices $V_1, V_2, ..., V_n, V_{n+1}$; a *free of cycles directed path* (just path from now on) is any sequence of arcs $\in E$ of the following type: $\{(V_1, V_2), (V_2, V_3), ..., (V_n, V_{n+1}))\}$.

A virtual network is represented by the undirected graph $G^k(V^k, E^k)$, where k is the identifier of the VN. Each virtual network request is represented by the graph $G^k(V^k, E^k)$, a group of variables $h(i^k, j^k)$ representing the bandwidth demand of the virtual link $(i^k, j^k) \in E^k$, a variable $cpu(i^k)$ representing the CPU requirement of the virtual node $i^k \in V^k$ and a variable $I(i^k, j^k)$ which represents the CPU demand of every intermediate node composing the substrate path that maps the virtual link (i^k, j^k); this value can be given by the virtual network requester, or can be calculated based on the bandwidth demand $h(i^k, j^k)$, Fig. 2 is an example of these requests over a substrate network with the NSF topology.

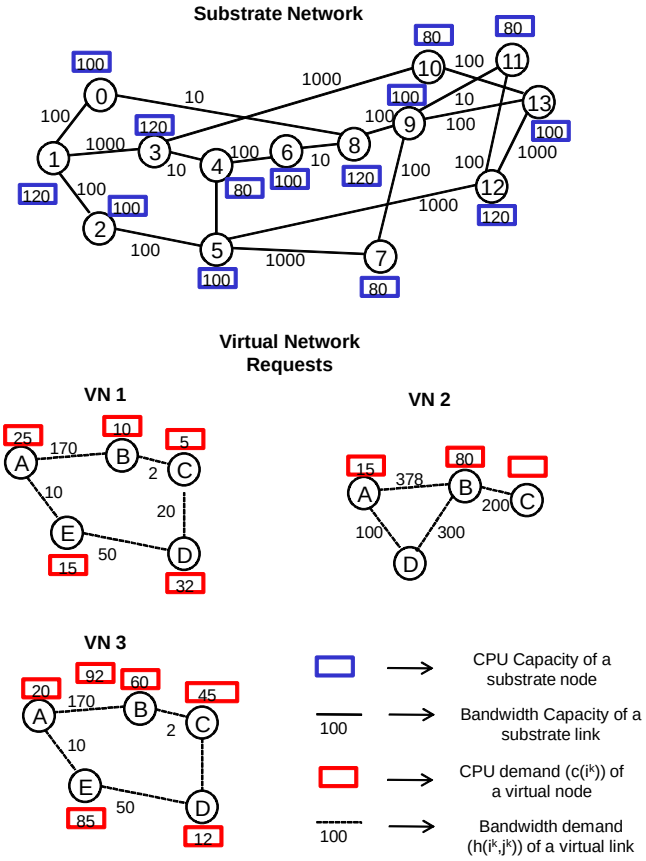

Fig. 2 Example of a Set of VN requests over a NSF Substrate Network

If $h(i^k, j^k) = 0$ the virtual link (i^k, j^k) is not asking for a specific bandwidth demand, in the same way if $cpu(i^k) = 0$ the virtual node i^k is not asking for a specific CPU demand. The notation of the substrate network variables is shown in Table 1, while variables of each specific virtual network are deeply detailed in Table 2.

Table 1 Definition of Substrate Network's Variables

Terms	Definition
$G(V, E)$	Undirected graph representing the substrate network
V	Set of physical nodes (routers) belonging to the substrate network
E	Set of links belonging to the substrate network
(i, j)	$(i, j) \in E$ is the link from substrate's node i to substrate's node j
VN	Set of virtual networks, virtualized from the substrate network
VN_k	$VN_k \in VN$ represents the virtual network number k
$C(i)$	CPU capacity of the substrate network's node i
$B(i, j)$	Bandwidth capacity of the substrate network's link (i, j)

Table 2 Definition of Variables Specific of Virtual Network k

Terms	Definition
$G^k(V^k, E^k)$	Undirected graph representing the virtual network k
V^k	Set of virtual nodes (routers) belonging to the virtual network k
	a virtual node mapping is given by the following function $f(i^k) \in V$ and $i^k \in V^k$, $f(i^k)$ has to fulfill the CPU demand and location constraints
$cpu(i^k)$	CPU that must be assigned to the node $i^k \in V^k$ of the virtual network k
E^k	Set of links belonging to the virtual network k
$P(i^k, j^k)$	Set of cycle-free paths in SN connecting the substrate nodes $f(i^k)$ and $f(j^k)$
$I(i^k, j^k)$	CPU requirement of a substrate node acting as intermediate (hidden hop) in a substrate path representing the virtual link (i^k, j^k)
$d^p(i)$	Assigned bandwidth to substrate node $i \in V$ acting as an intermediate node in path $p \in P(i^k, j^k)$
(i^k, j^k)	$(i^k, j^k) \in E^k$ is the link from virtual node i^k to node j^k in the virtual network k.
	This virtual link is given by the following function $g(i^k, j^k) \in P(i^k, j^k)$ and $(i^k, j^k) \in E^k$, $g(i^k, j^k)$ has to fulfill the bandwidth demand
$h(i^k, j^k)$	Bandwidth that must be assigned (demand) to the virtual link (i^k, j^k) of the virtual network k
$b^p(i^k, j^k)$	Allocated Bandwidth to the virtual link (i^k, j^k) in the path $p \in P(i^k, j^k)$ of the virtual network k
$\rho^p(i, j, k)$	Binary variable It is 0 → if the substrate link (i, j) in the virtual network k, is not part of the path p for the virtual link (i^k, j^k). It is 1 elsewhere
$\alpha^p(i^k, j^k, l)$	Binary variable It is 0 → if the substrate node $l \in V$ in the virtual network k, is not an intermediate node of the path p for the virtual link (i^k, j^k). It is 1 elsewhere

2.3 Optimization Model Definition

The model aims to maximize the spare bandwidth and CPU in the physical network, while the demands for virtual nodes and links are fulfilled. Formally the optimization model is the following:

Maximize:

$$F = \sum_{(i,j)\in E} \left(B(i,j) - \sum_{k=1}^{|VN|} \sum_{(i^k,j^k)\in E^k} \sum_{p=1}^{|P(i^k,j^k)|} \rho^p(i,j,k) b^p(i^k,j^k) \right) + \sum_{l\in V} \left(C(l) - \sum_{k=1}^{|VN|} \left(\sum_{i^k \in V^k | f(i^k)=l} cpu(i^k) + + \sum_{(i^k,j^k)\in E^k} \sum_{p=1}^{|P(i^k,j^k)|} \alpha^p(i^k,j^k,l) d^p(l) \right) \right) \tag{1}$$

Subject to:

$$B(i,j) - \sum_{k=1}^{|VN|} \sum_{(i^k,j^k)\in E^k} \sum_{p=1}^{|P(i^k,j^k)|} \rho^p(i,j,k) b^p(i^k,j^k) \geq 0, \forall (i,j) \in E \tag{2}$$

$$C(l) - \sum_{k=1}^{|VN|} \left(\sum_{i^k \in V^k | f(i^k)=l} cpu(i^k) + + \sum_{(i^k,j^k)\in E^k} \sum_{p=1}^{|P(i^k,j^k)|} \alpha^p(i^k,j^k,l) d^p(l) \right) \geq 0, \quad \forall l \in V \tag{3}$$

$$\sum_{p=1}^{|P(i^k,j^k)|} \rho^p(i,j,k) b^p(i^k,j^k) = h(i^k,j^k) \quad for \quad 1 \leq k \leq |VN|, \forall (i^k,j^k) \in E^k \tag{4}$$

$$\sum_{p=1}^{|P(i^k,j^k)|} \alpha^p(i^k,j^k,l) d^p(l) = I(i^k,j^k) \quad \forall l \in V, \quad for \quad 1 \leq k \leq |VN| \quad and \quad \forall (i^k,j^k) \in E^k \tag{5}$$

$$\exists! b^p(i^k,j^k) \neq 0, \quad \forall p \in P(i^k,j^k) \tag{6}$$

$$\exists! d^p(n) \neq 0, \quad \forall p | p \in P(i^k,j^k) \tag{7}$$

$$\begin{aligned} |VN| \geq 0 \quad |E| \geq 0 \\ |P(i^k,j^k)| \geq 0 \quad for \quad 1 \leq k \leq |VN|, 1 \leq l \leq |C(k)| \\ \rho^p(i,j,k) \geq 0 \quad for \quad (i,j) \in E, 1 \leq k \leq |VN|, 1 \leq p \leq |P_l(k)| \\ \alpha^p(i^k,j^k,l) \geq 0 \quad for \quad (i^k,j^k) \in E^k, l \in V, 1 \leq k \leq |VN|, \\ 1 \leq p \leq |P_l(k)| \end{aligned} \tag{8}$$

The objective is to find, for each virtual network, the paths accomplishing the constraints that maximize the objective function F (Equation (1)), representing the sum of the spare bandwidth and spare CPU in the substrate network. First group of constraints (2), (3) are of capacity, they assure the sum of the bandwidths and CPU, assigned to each virtual link and node, does not exceed substrate's link nor node capacity. Second

group of constraints are related with demand (4), (5); they assure that each virtual link obeys its demanded bandwidth and each intermediate virtual node reaches its CPU demand. Third set of constraints (6), (7) assures the uniqueness of the chosen substrate path for each virtual link. Last set of constraints (9) force variables to be greater or equal than zero.

3 Problem Complexity

The model exposed in the previous section may have different degrees of complexity depending on the way the allocation of the virtual link demands is approached. To consider unique substrate path to transport a virtual link demand is often a realistic restriction due to the used routing protocol, or simply an explicit management requirement stipulating avoidance of packet re-sequencing in receiving nodes. In the right side's top of Fig. 2 an example of an allocation using just one path is shown. If this restriction is present in the problem, it can be formulated as the unsplittable flow decision problem (UFP) [9]:

INSTANCE: Graph $G = (V, E)$, and edge capacity $B(i, j)$ for each $(i, j) \in E$, a set of demands T, where $h(s, t)) \in T$ is the demand between nodes s and $t \in V$. We say that T is realizable in G if there exist a set of paths $P(s, t)$ such that s and t are the endpoints of each path, and the following capacity constraint is met for every edge:

$$\sum_{(s,t)\in T} \sum_{p\in P(s,t)} \rho^p(i,j)h(s,t) \leq B(i,j) \quad \forall (i,j) \in E \quad (9)$$

where $\rho^p(i, j)$ is a binary variable that is 1 if the link $(i, j) \in E$ is part of the path p between the nodes (s, t), and 0 otherwise.

QUESTION: Is T realizable in G?

This problem is a generalization of the well known Edge Disjoint Paths (EDP) decision problem, that was shown to be NP-complete [6], that is why UFP is NP-complete.

Fig. 3 shows a virtual network on top of a substrate. There is a virtual network demand between virtual nodes A and D. In the right side of the figure; two methods to allocate the traffic are shown, in part a, the bandwidth to fulfill the virtual link demand is assigned to just one of the possible paths between this pair of nodes. In the bottom of the figure (part b), the allocation of bandwidth is made using all the possible paths between the substrate nodes. This approach can be solved using the multi-commodity-flow problem (MFP). Algorithms running in polynomial time that solve MFP are available [1].

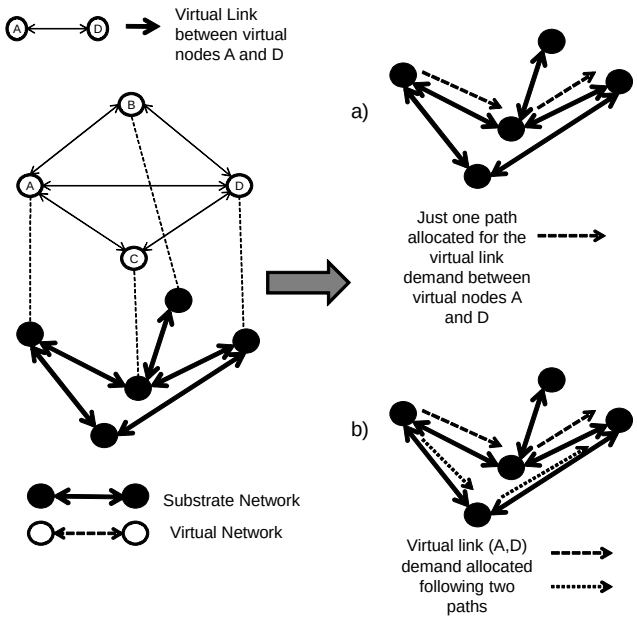

Fig. 3 Different Strategies to Allocate Virtual Link Demands

4 Heuristics

In this section, we propose an algorithm to map a set of virtual network requests (offline algorithm) trying to maximize the spare bandwidth and spare CPU in the substrate network. The algorithm takes into account that each VL request mapped to a SN path uses certain CPU capacity in the intermediate path nodes; this capacity could be calculated as a function of the virtual link BW demand as well as a given parameter enclosed in this demand.

The algorithm is divided in two different steps: firstly, the algorithm maps the requests of each VN, both for the virtual nodes and virtual links, that explicitly ask their demands. In second place, the remaining resources are distributed equally among the remaining virtual nodes and links.

The second step of the algorithm, to allocate the resources equally among the request without demand, is divided in two parts; the virtual node mapping is easy because we assume that each virtual node is already mapped ($f(i^k)$ is known for $i^k \in V^k \forall k$), the mapping of a virtual link is performed by allocating the resources to the shortest SN path.

The optimum solution to the first step of the algorithm is computationally hard to solve as shown in the previous section, so a heuristic solution is proposed

to find a near-optimum result. This heuristic is mainly based in an approximated greedy algorithm [15] proposed to solve the Unsplittable-Flow problem, we make few modification to adapt the algorithm to the mapping problem.

4.1 Mapping of Demanded Virtual Nodes and Links

The algorithm to map the virtual nodes and links, of all the virtual network requests asking for explicit demand, uses a sub-algorithm to realize just one virtual link mapping. For the mapping of virtual nodes, the assumption that virtual nodes are already mapped is done, that is, each VN k request provides $f(i^k) \quad \forall i^k \in V^k$.

We employ a small modification of the known greedy algorithm (GA) [9] to look for the best substrate virtual path for a single virtual link obeying the bandwidth and CPU constraints. This algorithm uses the shortest path, with hops number as metric, as the preferred substrate path to map each virtual link. This decision is made because when the shortest path is used, the spare bandwidth and CPU of the SN is maximized, that is, as the shortest path maps the required demands to less resources (less physical links and nodes in the shortest path), the remaining resources in the network are maximized each time a shortest path is used.

Before presenting the algorithm that allocates a substrate path for each virtual link, some temporary variables and functions are explained in detail.

The algorithm needs some entry parameters: the graph $G(V, E)$ representing the substrate network, as well as the virtual nodes of the virtual link (i^k, j^k) that have associated the bandwidth demand $h(i^k, j^k)$ and CPU demand $I(i^k, j^k)$ $\forall i \in V$ acting as an intermediate node belonging to the path between $f(i^k)$ and $f(j^k)$. The function FIND-SP $(n, i^k, j^k, G(V, E))$ returns the shortest path (in number of hops) number n between the nodes $f(i^k)$ and $f(j^k) \in V$ accomplishing the CPU and Bandwidth restrictions, this function is based on the algorithm to find the k-shortest paths proposed in [5]. HOPS(p) is a function that returns the number of hops of a path p. MAX(Paths, bin) returns the maximum from a set of paths taking into account the bin variable that can be BW or CPU, that is, if bin=BW the chosen path will be the path with more remaining bandwidth (sum of the remaining BW in all links). If two (or more) paths have the same remaining bandwidth, the one with higher spare CPU (sum of the remaining CPU in all nodes of the path) will be chosen. If bin=CPU the opposite process is made. INTER(p) returns a set containing the intermediate nodes of the path p.

Algorithm 1 Greedy Algorithm to Map a Single Virtual Link (GA)

Require: $G(V, E)$, i^k, j^k, bin
Condition=TRUE, FT=TRUE, SP={∅}, paths={∅}, Counter=1, prevhops=0, Path
while Condition **do**
    SP=FIND-SP(Counter,i^k,j^k,$G(V, E)$)
    if HOPS(SP)≠0 **then**
        if HOPS(SP) = prevhops or FT **then**
            FT=FALSE, paths=paths+{SP}, prevhops=HOPS(SP)
        else
            Condition=FALSE
        end if
    else
        Condition=FALSE
    end if
end while
if HOPS(SP)≠ {∅} **then**
    Path=MAX(paths,bin)
    for $i \in INTER(Path)$ **do**
        $C(i) = C(i) - I(i^k, j^k)$
    end for
    return Path
else
    return 0
end if

The algorithm GA (Alg. 1) works by finding the shortest path, with number of hops as the metric, accomplishing the bandwidth and CPU constraints. If more than one shortest path is found, a path with more remaining bandwidth or remaining CPU, based on the bin variable, is chosen. When the path is chosen, the algorithm proceeds to subtract the intermediate node demand associated with the virtual link from the remaining CPU demand of each intermediate node.

The general algorithm (GAP-M) maps the virtual links and nodes with explicit demands of one virtual network request making use of the previous procedure (GA) and uses the following temporary variables and functions.

H and A are sets of unmapped and mapped (respectively) connections of a specific virtual network. SORT(B) returns the group of links $\in B$ ordered in decreasing order of demands $h(i^k, j^k)$. FIRST(H) returns the first item $(i^k, j^k) \in E^k$ of H. SORTP(A) returns the connections of A in decreasing order, taking into account the number of hops of each corresponding SN path. REALLOCATE-RESOURCES(k) updates the capacities of the substrate network ($B(i, j)$, $\forall (i, j) \in E$ and $C(i), \forall i \in V$) by reallocating them with the mapped virtual links and nodes resources of the virtual network k. The variable TOL (tolerance) is the maximum number of times that the largest path is eliminated to allow the mapping of a request, that does not fit in the physical network, due to the lack of resources.

Algorithm 2 Algorithm to Map a Virtual Network (GAP-M)

Require: $G(V^k, E^k)$, $G(V, E)$, TOL, bin
1: **for** Counter $\in V^k$ **do**
2: **if** $C(f(counter)) \geq cpu(counter)$ **then**
3: C(f(counter))= C(f(counter)) - cpu(counter)
4: **else**
5: GOTO 39
6: **end if**
7: **end for**
8: Path=0, A=$\{\emptyset\}$, B=E^k
9: H=SORT(B), (i^k, j^k)=FIRST(H), Path=GA(G(V,E), $f(i^k)$,$f(j^k)$, bin)
10: **if** Path$\neq$ 0 **then**
11: $g(i^k, j^k)$=Path
12: **for** $(i, j) \in g(i^k, j^k)$ **do**
13: $B(i, j) = B(i, j) - h(i^k, j^k)$
14: **end for**
15: H=H-$\{(i^k, j^k)\}$, A=A+$\{(i^k, j^k), Path\}$, GOTO 9
16: **else**
17: **if** H=$\{\emptyset\}$ **then**
18: Mapping completed, Stop Algorithm
19: **else**
20: GOTO 22
21: **end if**
22: **end if**
23: **for** Counter=1 to TOL **do**
24: D= SORTP(A), (g^k, h^k)=FIRST(D)
25: H=H+$\{(g^k, h^k)\}$, A=A-$\{(g^k, h^k), Path\}$
26: **for** $(i, j) \in g(g^k, h^k)$ **do**
27: $B(i, j) = B(i, j) + h(i^k, j^k)$
28: **end for**
29: Path=GA(G(V,E), $f(i^k)$,$f(j^k)$, bin)
30: **if** Path$\neq$ 0 **then**
31: $g(i^k, j^k)$=Path
32: **for** $(i, j) \in g(i^k, j^k)$ **do**
33: $B(i, j) = B(i, j) - h(i^k, j^k)$
34: **end for**
35: H=H-$\{(i^k, j^k)\}$, A=A+$\{(i^k, j^k), Path\}$, B=H, GOTO 9
36: **end if**
37: **end for**
38: **if** Path=0 **then**
39: It is not possible to map Virtual Network k
40: REALLOCATE-RESOURCES(k)
41: **return** 0
42: **end if**

Algorithm GAP-M works by mapping each virtual link, belonging to the virtual network. Virtual links are mapped, one by one, in decreasing demand order. To map each link, the algorithm GA (Alg. 1) is used. If the mapping of a virtual link is not possible (GA returns 0), the algorithm tries to look for the resources needed to assign this requirement; the longest of the mapped paths (already assigned to another virtual link) is eliminated of the mapping because this path is more likely to share substrate link and nodes with the current request, and so, the resources are returned to the substrate network variable and another try to map virtual link is done. If even in this way the request can not be mapped, the following mapped longest path is eliminated and the same process is performed until the virtual link is assigned or until a tolerance (TOL) value of times is reached. This value is an algorithm input and is thought to limit the number of mapping attempts of a virtual link; if after TOL attempts, the mapping has not been done, we consider that this virtual network can not be mapped.

To map the set of VNs into the SN, the previous algorithm (Alg. 2) must be called for each VN.

4.2 Mapping of Undemanded Virtual Nodes and Links

After the mapping of the request asking for a specific demand, the second step in the algorithm is to assign the remaining unconstrained demand in an equal way among the requests that did not demand any resource. To do the mapping, it is necessary to know, in first place, if the physical nodes that correspond to the virtual nodes (without demand), have non-zero CPU remaining capacity. To map a virtual link, the shortest path with non-zero remaining capacity, both in virtual links and nodes, is chosen.

The algorithm uses a new function SP-WR ($G(V, E)$, i, j) that returns the shortest path between i and j in the graph $G(V, E)$, this path must have non-zero remaining bandwidth and CPU in each physical link (composing the virtual one) and physical intermediate node, if this condition does not apply the function returns 0.

Algorithm 3 Algorithm to Map Request Without Demand in a Virtual Network (UVL)

Require: $G(V^k, E^k)$, $G(V, E)$
1: **for** Counter $\in V^k$ **do**
2: **if** $C(f(counter)) = 0$) **then**
3: GOTO 16
4: **end if**
5: **end for**
6: U=E^k
7: (i^k, j^k)=FIRST(U), Path=SP-WR($G(V, E), f(i^k), f(j^k)$)
8: **if** Path$\neq$ 0 **then**
9: $g(i^k, j^k)$=Path, U=U-$\{(i^k, j^k)\}$, GOTO 7
10: **else**
11: **if** H=$\{\emptyset\}$ **then**
12: Mapping completed, Stop Algorithm
13: **end if**
14: **end if**
15: **if** Path=0 **then**
16: It is not possible to map Virtual Network k
17: **return** 0
18: **end if**

The mapping process finishes with the execution of the UVL algorithm (Alg. 3) in each virtual network request. In first place, the algorithm assures that each

physical node corresponding with a virtual node of this request, has remaining CPU capacity, if it is not the case, the virtual network is not mapped. After that step, the virtual link mapping is performed, the result of the mapping is located in $g(i^k, j^k)$, these variables indicates the path each virtual link has been mapped to.

5 Example of Application

Fig. 4 shows an example with a simple SN topology and three virtual network requests, two of them with demands in each virtual node and link, the third request does not ask for any demand. This example is presented to show how a network would behave following the proposed algorithm. It will be easier for the reader to understand how the mapping is carried out.

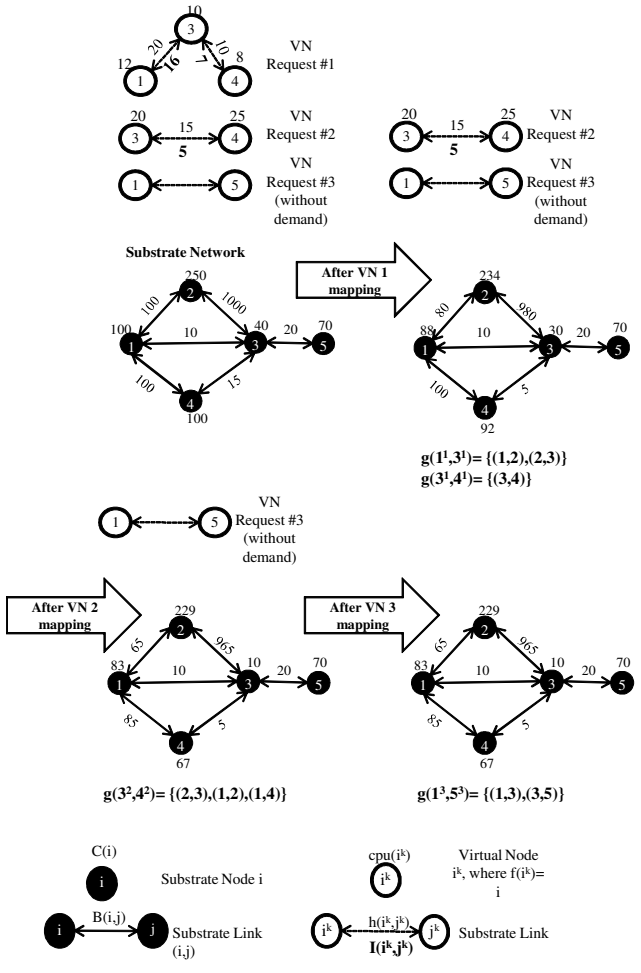

Fig. 4 Example of a Virtual Network Mapping Scenario

In this example, three virtual network request are done to a substrate network that counts on 5 nodes connected by 6 links. The information of each physical node (node id i, remaining bandwidth capacity $C(i)$) and physical link (link remaining capacity, $B(i,j)$) is shown in the figure. The virtual network requests contains also all the needed information in each VN. It is possible to see, for each virtual node being part of VN_k, in Fig. 4; the node i to which each virtual node i^k is mapped (to facilitate the reading, $f(i^k) = i$) and the CPU demand $cpu(i^k)$, and for each virtual link, the bandwidth demand $h(i^k, j^k)$ and the CPU requirement of the intermediate physical nodes $I(i^k, j^k)$ that will compose the SN path fulfilling the virtual link requirements.

Following the proposed algorithms, the process to map the VN requests is as follows: Firstly, the 1st request is processed by GAP-M (Alg. 2), lets suppose arbitrary values to $TOL = 2$ and bin=BW. The first step of the algorithm is to subtract of the remaining CPU in physical nodes, the amount of demand made by the corresponding virtual nodes. The result is displayed in the drew SN after the VN 1 mapping. Nodes 1, 3 and 4 (nodes of the 1st VN request) new CPU capacity is the result of the subtraction of the virtual node demand from the old capacity. Node 2 remaining CPU decreases as well. In second place, the algorithm looks for the virtual link request with the higher bandwidth demand ($h(i^k, j^k)$), the result is the virtual link $(1, 3)$ with 20 units of bandwidth demand. In third place, a proper path fulfilling the virtual link requirements is searched, to find this path the GA (1) is invoked, this algorithm returns a path accomplishing the bandwidth and intermediate node CPU requirements. In this case, the chosen path is $g(1^1, 3^1) = \{(1, 2), (2, 3)\}$, although the shortest path between nodes 1 and 3 is direct (no hops), it does not obey the BW requirements. So the demanded bandwidth is subtracted from the remaining bandwidth capacity of each physical link contained in the substrate path. As this path contains an intermediate node (node 2), the CPU demand associated with the bandwidth request is subtracted from the remaining CPU capacity in this physical node. The case of the second virtual link in the first request is easier to solve. The shortest path between physical nodes 3 and 4 can accomplish the bandwidth and intermediate nodes CPU' demand, so the bandwidth demand $h(3^1, 4^1)$ is subtracted from the remaining bandwidth in this link $B(3, 4)$ and it is clear that $g(3^1, 4^1) = \{(3, 4)\}$. The result of this mapping is the substrate network shown in Fig. 4 after "VN 1 mapping" arrow.

After the first virtual network is mapped, the same algorithm is applied to the following VN request. The 2nd request is just of one link: $(3^2, 4^2)$. So, as in the previous request, the first step is to update the remaining capacities of the physical node. In this case,

nodes 3 and 4 remaining CPU is updated. The second step, with GA algorithm's help, is to find the most appropriate path for the unique virtual link request of the request. Possible paths between these nodes are $(\{(3,4)\},\{(1,3),(1,4)\},\{(2,3),(1,2),\ (1,4)\})$, but the only path obeying the bandwidth and CPU requirements is the largest $g(3^2,4^2)=\{(2,3),(1,2),(1,4)\}$, so this is the chosen path to map the virtual link $(3^2,4^2)$, and its bandwidth demand $(h(3^2,4^2))$ is subtracted from the bandwidth remaining capacity $(B(i,j))$ of each substrate link composing the path. The CPU requirement associated with the virtual link $I(3^2,4^2)$ is also subtracted from the remaining CPU capacity of each intermediate node (nodes 2 and 1). After the successful mapping of the VN 2 request, the resultant substrate network is shown in Fig. 4 after "VN 2 mapping" arrow.

The last VN request does not ask for any demand neither of virtual nodes nor of virtual links. So, as it is shown in Fig. 4 the substrate network does not change after the mapping of this request, the remaining resources are available to be used by this VN. But it is still interesting to know which is substrate path that will be chosen to fulfill the demand of the unique virtual link in the third VN request. This request among the substrate nodes 1 and 5 is fulfilled, as it is stated in the algorithm Alg. 3, using the shortest substrate path, with non-zero remaining resources (BW and CPU), among these nodes; in this case it is clear that the chosen path is $g(3^2,4^2)=\{(2,3),(1,2)\}$.

6 Conclusions and Future Work

The introduction of network virtualization would produce big changes in current network architecture. But it sets out some problems, such as the mapping of virtual networks in top of the substrate network.

We have proposed an optimization model contemplating the possibility of mapping virtual networks with explicit bandwidth and CPU demands, but also with the possibility of accept virtual links or nodes, or even virtual networks, that do not ask for any demand. As the problem is computationally hard to solve, we have proposed an algorithm (heuristic) that tries to map in first place, the virtual links and nodes that are asking for an explicit demand in each virtual network, this mapping is done with the objective of maximize the spare bandwidth and CPU resources in the substrate network; the aim of this optimization objective is to leave as much resources as possible, but obeying all the demands, to distribute them among the remaining requests (without explicit demands).

Previous studies in the embedding problem have been focused in the mapping of VNs taking into account the demand in both, virtual link bandwidth and virtual node CPU. CPU demand due the mapping of a virtual links in the intermediate nodes (hidden hops) of a substrate path, has not been taken into account as an important parameter in the problem. It is logical to think that some CPU is consumed by the intermediate nodes, because they will have to be configured to process and forward the packets passing through that virtual link. The proposed optimization model (and so the algorithm) includes this important feature.

The proposed algorithm is based on a modified greedy heuristic that can be improved, if the problem is set out as a Multi-Commodity flow problem, and after solving it, the application of rounding techniques might drive us to more optimized solution taking into account the constraint stating that just one SN path must be used to map each virtual link.

An optimization objective that is energy-awareness could be also interesting to obtain an embedding that reduces the energy expense in the substrate network.

Acknowledgements This work has been partially supported by national Spanish project CICYT TSI2007-66637-C02, European Network of Excellence Euro-NF FP7-ICT-2007-1, Grant No. 216366, AutoI: STREP, FP7 Call1 (ICT-2007-1-216404) and ResumeNet STREP, FP7 Call2 (ICT-2007-2-224619). Also, it has been partially supported by the "Comissionat per a Universitats i Recerca del DIUE" from the "Generalitat de Catalunya" and the Social European Budget ("Fons Social Europeu").

References

1. R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows*. Englewood Cliffs: Prentice Hall, 1993.
2. A. Berl, A. Fischer, and H. de Meer. Using system virtualization to create virtualized networks. In *Workshops der Wissenschaftlichen Konferenz Kommunikation in Verteilten Systemen (WowKiVS2009)*, Kassel, Germany, March 2009. vol. 17, EASST.
3. N. M. M. K. Chowdhury. Network virtualization: State of the art and research challenges. *IEEE Communications Magazine*, 47(7):20–26, July 2009.
4. N. M. M. K. Chowdhury, Muntasir Raihnan Rahman, and Raouf Boutaba. Virtual network embedding with coordinated node and link mapping. In *Proc. IEEE INFOCOM*. IEEE Infocom, April 2009.
5. D. Eppstein. Finding the k shortest paths. *SIAM Journal on Computing*, 28(2):652–673, 1998.
6. M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, New York, 1979.
7. X. Hesselbach and Ramon Fabregat. The impact over the packets sequence at the output interface in load balancing strategies. In *International Conference on Transparent Optical Networks*, volume 3, pages 263–266, 2006.
8. S. Kent and K. Seo. Security architecture for the internet protocol, December 2005. RFC 4301.
9. J.M. Kleinberg. *Aproximation Algorithms for Disjoint Paths Problems*. PhD thesis, Massachusetts Institute of Technology, 1996.

10. J. Lu and J. Turner. Efficient mapping of virtual networks onto a shared substrate. Technical Report WUCSE-2006-35, Washington University, 2006.
11. P. Papadimitriou, O. Maennel, A. Greenhalgh, A. Feldmann, and Laurent Mathy. Implementing network virtualization for a future internet. In *20th ITC Specialist Seminar*, Hoi An, Vietnam, May 2009.
12. L. Peterson, S. Shenker, J. Turner, et al. Overcoming the internet impasse through virtualization. *IEEE Computer*, 38(4):34–41, April 2005.
13. Ralf Steinmetz and Klaus Wehrle. *Peer-to-Peer Systems and Applications (Lecture Notes in Computer Science)*. Springer-Verlag New York, Secaucus, NJ, USA, 2005.
14. K. Tutschku, T. Zinner, A. Nakao, and P. Tran-Gia. Network virtualization: Implementation steps towards the future internet. In *Proc. of the Workshop on Overlay and Network Virtualization at KiVS*, Kassel, Germany, March 2009.
15. Krzysztof Walkowiak. New algorithms for the unsplittable flow problem. *Lecture Notes in Computer Science*, 3981:1101–1110, 2006.
16. Minlan Yu, Yung Yi, Jennifer Rexford, and Mung Chiang. Rethinking virtual network embedding: Substrate support for path splitting and migration. *ACM SIGCOMM CCR*, 38(2):17–29, April 2008.
17. Y. Zhu and M. Ammar. Algorithms for assigning substrate network resources to virtual network components. In *Proc. IEEE INFOCOM*, pages 2812–2823, April 2006.