A Virtual Network Mapping Algorithm based on Subgraph Isomorphism Detection

Jens Lischka, Holger Karl
Paderborn Center for Parallel Computing
Paderborn University
33102 Paderborn, Germany
jeli@mail.uni-paderborn.de, holger.karl@mail.uni-paderborn.de

ABSTRACT

Assigning the resources of a virtual network to the components of a physical network, called Virtual Network Mapping, plays a central role in network virtualization. Existing approaches use classical heuristics like simulated annealing or attempt a two stage solution by solving the node mapping in a first stage and doing the link mapping in a second stage.

The contribution of this paper is a Virtual Network Mapping (VNM) algorithm based on subgraph isomorphism detection: it maps nodes and links during the same stage. Our experimental evaluations show that this method results in better mappings and is faster than the two stage approach, especially for large virtual networks with high resource consumption which are hard to map.

Categories and Subject Descriptors

C2.5 [**Computer-Communication Networks**]: Local and Wide-Area Networks; G.1.6 [**Numerical Analysis**]: Optimization

General Terms

Algorithms

Keywords

Virtual Network Mapping; Network Embedding; Resource Allocation; Subgraph Isomorphism Detection; Network Virtualization

1. INTRODUCTION

Virtualization is a well investigated research area in computer science. One of its initial purposes is to run multiple different applications (e.g. servers, operating systems) upon the same shared physical resources. Network Virtualization has become more and more important over the last years. It is used for example for network simulation [10, 1, 8] or to provide customized end-to-end services over the same physical network [6, 14].

Virtual Network Mapping (VNM) plays a central role in building a virtual network (VN). During this mapping process each node of the VN is assigned to a node of the physical network (PN) and each virtual link is assigned to a path or flow in the PN, such that a set of previously defined constraints (e.g. topology constraints, data rate, CPU capacity) is satisfied.

The main objective in solving the Virtual Network Mapping Problem (VNMP) is to make efficient use of the underlying resources, while still satisfying the set of previously defined mapping constraints. In addition, a VNMP Algorithm should be able to handle dynamicly arriving online requests and also offer admission control, since some VN requests must be rejected or postponed to avoid violation of resource guarantees for already existing virtual networks [15].

Several efficient VNMP heuristics solving different variants of the VNMP have been proposed in the past years [11, 17, 13, 5, 9, 15]. Some try to solve the problem considering data rate constraints [5, 9] while others restrict the search space by only solving the link embedding, since they assume that the node mapping is known in advance [13]. Ref. [11] describes a simulated annealing approach to map VNs onto the Emulab [4] infrastructure, and Ref. [15] presents a two stage mapping algorithm, handling the node mapping in a first stage and doing the link mapping in a second stage, based on shortest path and multi commodity flow detection.

In contrast to existing approaches, in this paper we propose a backtracking algorithm based on a subgraph isomorphism search method [2] that maps nodes and links during the same stage. The advantage of this single stage approach is that link mapping constraints are taken into account at each step of the mapping. When a bad mapping decision is detected it can be revised by simply backtracking to the last valid mapping decision, whereas the two stage approach has to remap all links which is very expensive in terms of runtime. Our experimental evaluations show that our subgraph isomorphism based method results in better mappings and is faster than the traditional two stage approach.

The remainder of this paper is organized as follows. Section 2 introduces terms and definitions related to the VNMP we use throughout the rest of the paper. Section 3 then specifies the vnmFlib algorithm, which is a modified version of the Vflib graph matching algorithm. Section 4 presents some experimental results and a performance comparison with a two stage VNM algorithm, Section 5 concludes the paper.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
VISA'09, August 17, 2009, Barcelona, Spain.
Copyright 2009 ACM 978-1-60558-595-6/09/08 ...$10.00.

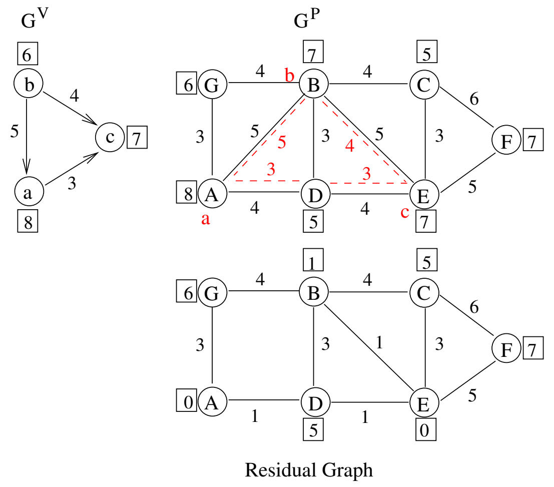

Figure 1: Example of virtual network mapping

2. VN MAPPING MODEL AND PROBLEM FORMULATION

The definitions presented in the following correspond to the problem formulation of [15].

Definition 1 (Network). *A Network is given by a directed graph $G = (N, L, C)$, where N is a set of nodes and L a set of links. Each node or link $e \in N \cup L$ is associated with a set of constraints $C(e) = \{C_1(e), ..., C_m(e)\}$.*

Definition 2 (Virtual Network Mapping (VNM)). *A VNM of a virtual network $G^V = (N^V, L^V, C^V)$ onto a physical network $G^P = (N^P, L^P, C^P)$ is defined as a mapping of G^V to a subset of G^P such that each virtual node is mapped onto exactly one physical node and each virtual link is mapped onto a loopfree path in the physical network:*

$$M : G^V \longrightarrow (N^P, P^P),$$

where P^P denotes the subset of all loopfree paths in G^P. M is called a valid *VNM if all constraints of G^V are satisfied and for each $l^V = (s^V, t^V) \in L^V$ there exists a path $p(s^P, t^P) \in P^P$ with $M(s^V) = s^P$ and $M(t^V) = t^P$. The VNM can be decomposed into node and link mapping as follows:*

$$\begin{array}{ll}\text{Node mapping :} & M_N : N^V \longrightarrow N^P \\ \text{Link mapping :} & M_L : L^V \longrightarrow P^P\end{array}$$

As an example consider G^V and G^P of Figure 1. Each node is associated with a CPU- and each link with a data rate constraint. The nodes a, b, c are mapped onto A, B, E and the virtual links are mapped to the paths $[A, D, E]$, $[B, A]$ and $[B, E]$. The mapping is valid, since the capacity constraints of the virtual network do not exceed the capacities of the physical network.

To define the costs of a VNM we consider a constraint cost function $cost_i(M(G^V))$ for each constraint $C_i \in C^V$. The costs of a VNM are given by the sum of the cost functions together with a tunable weight constant α_i for each C_i which allows to strike a balance between the different constraint costs:

Definition 3 (Virtual Network Mapping Costs). *Suppose a VN G^V with constraints $C^V = \{C_1^V, ..., C_m^V\}$, a PN G^P with constraints $C^P = \{C_1^P, ..., C_m^P\}$ and a VNM $M(G^V)$ of G^V onto G^P. The* total costs *of $M(G^V)$ are given by*

$$cost(M(G^V)) = \sum_{i=1}^{n} \alpha_i \cdot cost_i(M(G^V))$$

The definition of the constraint cost functions can vary and depends on the character of the corresponding constraint. In case of additive constraint costs like data rate or delay the constraint cost function could look like

$$cost_i(M(G^V)) = \sum_{l \in L^V} C_i^V(l) \cdot length(M(l)).$$

where $M(l)$ is the path in G^P to which the virtual link l is mapped and $length(M(l))$ is the length of the path. For multiplicative costs like error rate, the function would look like

$$cost_i(M(G^V)) = 1 - \sum_{l \in L^V} (C_i^V(l))^{length(M(l))}.$$

Consider the VNM of Figure 1. In addition to the constraints $C_1 =$CPU and $C_2 =$data rate we examine an $C_3 =$ error rate of 0.1 for each physical link. Further the costs are weighted equally with $\alpha_1 = \alpha_2 = \alpha_3 = 1$. The costs are $cost_1(M(G^V)) = 8 + 7 + 6 = 21$ for the node mapping and sum to $cost_2(M(G^V)) = 3{\cdot}2{+}4{\cdot}1{+}5{\cdot}1 = 15$ for the data rate and $cost_3(M(G^V)) = 1 - (0.1^2 + 0.1^1 + 0.1^1) = 0.79$ for the error rate. The total costs of the VNM are $cost(M(G^V) = 36.79$.

To handle multiple dynamically arriving mapping requests we next introduce the terms *Virtual Network Request* and *Residual Graph.*

Definition 4 (Virtual Network Request (VNR)). *A VNR $r_i = (G_i^V, a_i, l_i)$ consists of a virtual network G_i^V, an arrival time a_i and a life time l_i. The arrival time is the time a VNR should be mapped onto a PN and life time denotes the time period a VN should last on the PN.*

Definition 5 (Residual Graph). *Given a physical Network G^P, a virtual network G^V and a VNM of G^V onto G^P. We get the residual graph G_{res}^P of G^P by subtracting the capacities of each virtual node and link of G^V from the capacities of the physical nodes and links of G^P to which they are mapped.*

Again consider the mapping of Figure 1. Node a is mapped to node A and the residual CPU capacity is $C_1^P(A) - C_1^V(a) = 0$. The link $[a, c]$ is mapped to path $[A, D, E]$ which results in a residual capacity of 1 for the links $[A, D]$ and $[D, E]$.

3. THE ALGORITHM

The NP-complete Subgraph Isomorphism Detection problem [7] can be reduced to the VNM problem by assigning a single delay constraint of 1 to each physical and virtual link. The delay constraint is satisfied if the delay of a physical path does not exceed the delay of the virtual link that is mapped to it. A VNM of G^V onto G^P maps each node $n^V \in N^V$ to a node $n^P \in N^P$ and each virtual link $l^V \in L^V$ to a path p in P^P. To get a valid VNM the delay constraint has to be satisfied. That means that each l^V must be mapped onto a path of length ≤ 1 and therefore the resulting VNM is a subgraph isomorphism of G^V in G^P since

Algorithm 1 vnmFlib(G^V_{sub}, $M(G^V_{sub})$, G^V, G^P)

```
Require: a VN $G^V$, a PN $G^P$ and a subgraph $G^V_{sub}$ of $G^V$
 1: $C = \leftarrow$ genneigh($G^V_{sub}$, $G^V$, $G^P$)
 2: for each $(n^V, n^P)$ in C do
 3:   if valid(M($G^V_{sub}$), $(n^V, n^P)$, $G^P$) then
 4:     create $G^V_{sub}$ and $M(G^V_{sub})$ by adding $(n^V, n^P)$
 5:     vnmFlib($G^V_{sub}$, $M(G^V_{sub})$, $G^V$, $G^P_{res}$)
 6:   end if
 7:   if $G^V_{sub} == G^V$ then
 8:     return $M(G^V_{sub})$
 9:   end if
10: end for
```

each virtual link is mapped to exactly one physical link. Thus it seems to be a promising approach to use a subgraph isomorphism detection algorithm to solve the VNM problem.

The vnmFlib algorithm (Algorithm 1) described in this section is an extended version of the Vflib graph matching algorithm [2]. The main difference of vnmFlib and Vflib is that vnmFlib allows the mapping of links to paths shorter than a predefined distance value ϵ (in terms of hops), whereas Vflib is limited to link-on-link mappings. Note that if $\epsilon = 1$ the VNM of a VN G^V onto a PN G^P generated by vnmFlib is also a subgraph isomorphism of G^V in G^P. Another difference is that vnmFlib checks network constraints at each mapping step.

The algorithm tries to build a valid VNM solution by successively adding nodes and links of G^V to an initially empty subgraph G^V_{sub} of G^V. During the mapping process the algorithm ensures that $M(G^V_{sub})$ is a valid VNM of G^V_{sub} onto G^P. The algorithm terminates when G^V_{sub} fully covers G^V and returns $M(G^V_{sub})$, which is a valid VNM of G^V on G^P.

3.1 An Example

Figure 2 depicts the mapping process for the networks of Figure 1 with $\epsilon = 2$. In a first step a set C of node pairs (n^V, n^P) with $n^V \in N^V$ and $n^P \in N^P$ is generated by the *genneigh()* function (Algorithm 1 line 1). The algorithm adds n^V to G^V_{sub} and n^P to $M(G^V_{sub})$ respectively (Figure 2 (a)). Now the *valid()* function (line 3) checks whether the resulting mapping $M(G^V_{sub})$ is valid. If so the nodes are added to G^V_{sub} and $M(G^V_{sub})$ and vnmFlib is called with the corresponding residual graph G^P_{res} and the new subgraphs G^V_{sub} and $M(G^V_{sub})$ (line 5). Otherwise the termination condition is checked (line 7) and if it fails the next node pair of C is examined. Since the *valid()* function returns true for our example, vnmFlib is called with the newly created subgraph and mapping.

Again a vector of node pairs is generated and the first node pair (c, B) is added to G^V_{sub} and $M(G^V_{sub})$ (Figure 2 (b)). Now for all virtual links connecting the newly added node c with G^V_{sub} a path in G^P which satisfies all constraints has to be found and added to the mapping. For our example the only virtual link connecting node c with G^V_{sub} is $[a, c]$ and the algorithm adds path $[A, B]$ to the mapping, since it satisfies the data rate constraint. The mapping is valid and the algorithm proceeds with the generation of a new set of node pairs.

Next the algorithm maps node b to node G. This time the valid function fails because there exists no path from G to A

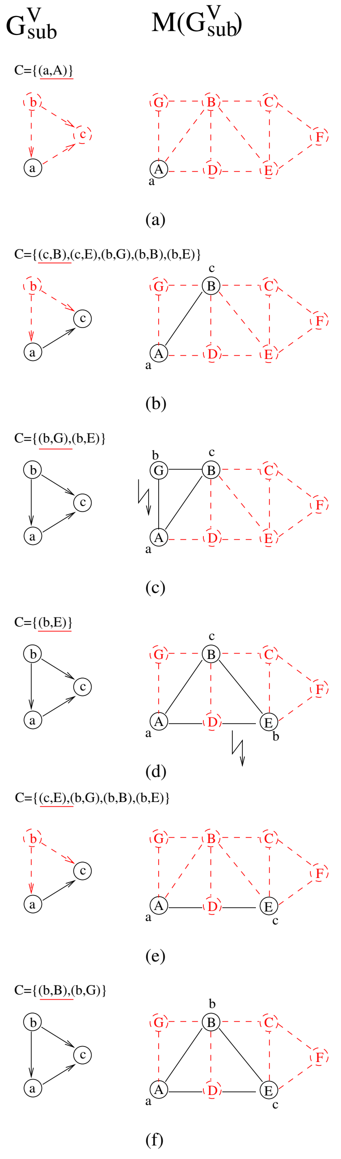

Figure 2: vnmFlib mapping process

in G^P with sufficient data rate capacity (Figure 2 (c)). Thus the algorithm tries to map the next pair (b, E) and fails for the same reason (Figure 2 (d)). As there is no node pair left in C the algorithm checks the termination condition (line 7), fails, does a backtracking step to the last valid mapping (Figure 2 (b)) and tries to map (c, E), which results in a valid mapping (Figure 2 (e)).

Again the algorithm computes a set of node pairs and maps the first one (Figure 2 (f)). This time the mapping is valid and the algorithm returns it (line 8) since G^V_{sub} fully covers G^V and the termination check of line 7 succeeds.

3.2 Algorithm Details

In the following we describe the two key functions of the vnmFlib algorithm, *genneigh()* and *valid()* in depth.

Before we can describe the procedure in more detail, we have to introduce the node set $F_{G_{sub}}(G)$:

Let $G = (N, L)$ be a directed graph and $G_{sub} = (N_{sub}, L_{sub})$

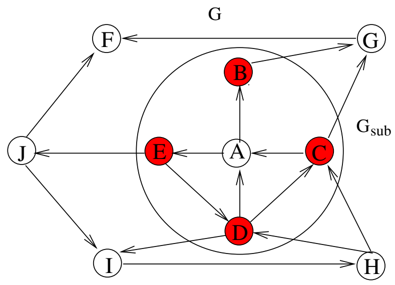

Figure 3: Set of front nodes $F_{G_{sub}}(G) = \{B, C, D, E\}$ of G_{sub} relative to G and $F_G(G_{sub}) = \{G, H, I, J\}$ respectively

Algorithm 2 genneigh(G^P, G^V, G^V_{sub}, $M(G^V_{sub})$)

```
Require: $G^P$, $G^V$, $G^V_{sub}$, $M(G^V_{sub})$
1: if $F_{G^V_{sub}}(G^V) = \emptyset$ then
2:     $C = N^V \times N^P$
3: else
4:     $C = F_{G^V_{sub}} \times G^P \setminus M_N(G^V_{sub})$
5: end if
6: optimize($C$)
7: sort($C$)
8: return($C$)
```

be a subgraph of G. The set

$$F_{G_{sub}}(G) = F^{in}_{G_{sub}}(G) \cup F^{out}_{G_{sub}}(G)$$

with

$$F^{in}_{G_{sub}}(G) = \{n_i | (n_i, n_j) \in L \wedge (n_j \in N_{sub} \wedge n_i \notin N_{sub}\}$$

and

$$F^{out}_{G_{sub}}(G) = \{n_j | (n_i, n_j) \in L \wedge (n_i \in N_{sub} \wedge n_j \notin N_{sub})\}$$

is called the *front* of G_{sub} in relation to G. Thus the front of G_{sub} in relation to G is the set of all nodes in G_{sub} that are adjacent to a link connecting G_{sub} with G.

As an example consider the graphs of Figure 3 with $N_{sub} = \{A, B, C, D, E\}$. The set of links starting in G and ending in G_{sub} are (H, D) and (H, C) and thus $F^{in}_{G_{sub}}(G) = \{C, D\}$. All links from G_{sub} to G are $(D, I), (E, J), (B, G)$ and (C, G). Consequently $F^{out}_{G_{sub}}(G)$ consists of the nodesets $\{B, C, D, E\}$ and $F_{G_{sub}}(G) = \{B, C, D, E\}$.

3.2.1 The genneigh() Function

The *genneigh()* function (Algorithm 2) takes four graphs as its input arguments: a VN G^V, a PN G^P, a subgraph G^V_{sub} of G^V and a VNM $M(G^V_{sub})$ of G^V_{sub} onto G^P which can also be seen as a graph. In a first step the function generates the front of G^V_{sub} relative to G^V and checks if it is empty. Again consider the example of Figure 2. In the initial step of the vnmFlib algorithm G^V_{sub} is empty and thus $F_{G^V_{sub}}(G^V)$ is empty too. The algorithm computes the candidates set C as the cartesian product of all nodes in G^V and all nodes in G^P (line 2). But as we can see in Figure 2 (a), C consists of only one node pair (a, A). The reduction of the node pairs is done in the *optimize()* function in line 6. The function deletes all node pairs from C which do not satisfy the CPU

Rule	Condition
R-node	True iff n^P satisfies all constraints C^V of n^V
R-pred	True iff for each predecessor n^V_p of n^V in G^V_{sub} there exists a path of length $\leq \epsilon$ from the corresponding node n^P_p in $M(G^V_{sub})$ to n^P that satisfies all constraints C^V of the virtual links $l = (n^V_p, n^V)$ in G^P.
R-succ	True iff for each successor n^V_s of n^V in G^V_{sub} there exists a path of length $\leq \epsilon$ from n^P in $M(G^V_{sub})$ to the corresponding node n^V_s that satisfies all constraints C^V_l of the virtual links $l = (n^V, n^V_s)$ in G^P.

Table 1: Validity check conditions

constraint. Since node a of G^V needs 8 CPU units and only node A of G^P can serve this request, all remaining node pairs are removed from C. Such optimizations can drastically reduce the search space of the algorithm and lead to better runtimes. Another optimization reducing the size of C is the restriction of the length of paths to which virtual links are mapped by the ϵ threshold. In the example of Figure 2 ϵ is set to 2 and thus only paths with length ≤ 2 are allowed. As a consequence in Figure 2 (b) mapping b onto F is forbidden.

A small ϵ-value can lead to better VNMs (in terms of mapping costs as described in Section 2) but could also increase the number of rejected VNs if it is chosen too restrictive. If $\epsilon = 1$, the algorithm tries to find a subgraph isomorphism of the VN in the PN. Such an isomorphism does often not exist, especially for larger VNs. In this case the vnmFlib algorithm would traverse the whole search tree which has worst case complexity $\Theta(|N^P|!|N^V|)$ [3]. To avoid this we introduce an upper bound ω on the number of mapping steps and force the algorithm to stop its search if ω is exceeded. Finding good ϵ and ω values is part of the evaluation process and described in Section 4.

In a last step the node pairs are sorted. The sorting criterion depends on the constraint set of the networks. It is usually better to map expensive nodes prior to cheaper nodes. By expensive nodes we mean nodes with a high resource consumption. For example the node a of the VN of Figure 1 has a CPU capacity of 8 and is thus more expensive than node b with capacity 6.

3.2.2 The valid() Function

The *valid()* function checks if the addition of node pairs (n^V, n^P) to a valid VNM again results in a valid VNM. The necessary rules which have to be checked are listed in Table 1.

The first rule R-node checks whether any node constraints are violated. In the example of Figure 1 CPU capacity is chosen as the only node constraint. Consider the addition of node pair (b, C). For this case the R-node rule would fail since C cannot satisfy the CPU capacity of b $(5 < 6)$ and *valid()* returns false.

The remaining rules R-pred and R-succ check for broken connections. As an example consider the addition of (c, B) in Figure 2 (b). The set of predecessors of c in G^V_{sub} is just a and the corresponding node in $M(G^V_{sub})$ is A. Because there exists a path from A to B in G^P with sufficient data rate and it is shorter than ϵ (length$([A, B]) = 1 \leq 2$)the rule succeeds. The R-succ rule does the same for all successors

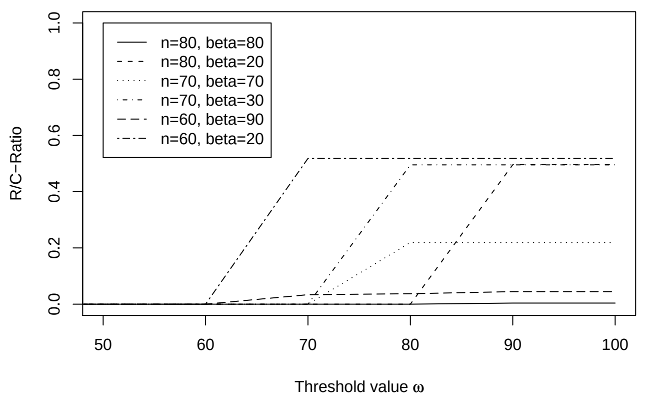

Figure 4: R/C-Ratio in relation to ω value for different n and β values.

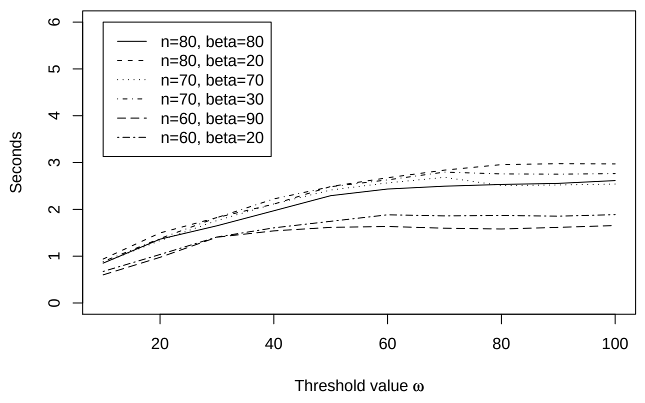

Figure 5: Runtime in relation to ω value for different n and β values.

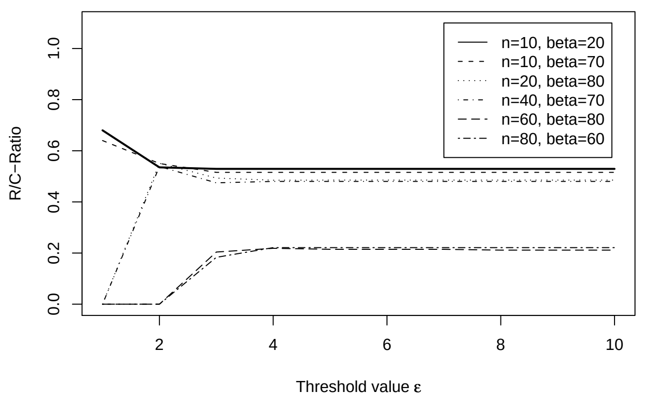

Figure 6: R/C-Ratio in relation to ϵ value for different n and β values.

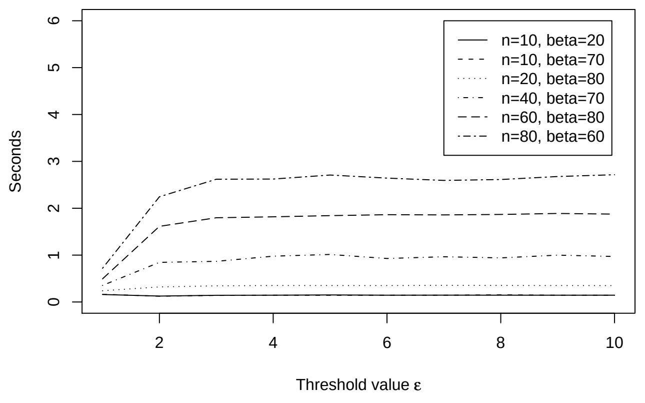

Figure 7: Runtime in relation to ϵ value for different n and β values.

respectively. All three conditions are checked beginning with the node rules. If one of the rules fails, *valid()* immediately stops and returns false, since the violation of one rule suffices to produce an invalid mapping.

4. EXPERIMENTAL RESULTS

In this section we first evaluate the runtime and output quality of the vnmFlib algorithm for different ϵ and ω values before we compare its performance to that of a two stage VNM algorithm [15].

4.1 Evaluation Environment

The networks for our experimental evaluations are associated with two Constraints C_1 and C_2, where $C_1(n)$ denotes the CPU-capacity of a node n and $C_2(l)$ the data rate capacity of a link l. The mapping costs (s. Section 2) are defined by the constraint cost functions $cost_1(M(G^V)) = \sum_{n \in N^V} C_1(n)$ and $cost_2(M(G^V)) = \sum_{l \in L^V} C_2(l) \cdot length(M(l))$ with $\alpha_1 = \alpha_2 = 1$. To express the quality of a VNM we introduce the *R/C-Ratio* (Revenue-to-Cost-Ratio) as the ratio of revenue to mapping costs $R(G^V)/cost(M(G^V))$ with the *Revenue* of a VN defined as

$$R(G^V) = \alpha_1 \cdot \sum_{n \in N^V} C_1(n) + \alpha_2 \cdot \sum_{l \in L^V} C_2(l)$$

If a VN cannot be mapped the R/C-Ratio is set to 0. Note that the R/C-Ratio takes on values between 0 and 1, where 1 is an optimal VNM.

Network Setup. We used the GT-ITM tool [16] to generate the physical topologies. Like in [15] the physical networks are configured to have 100 nodes and around 500 links, a scale that corresponds to a medium-size ISP. The CPU constraints at nodes and the link data rates follow a uniform distribution from 0 to 100 units.

The size n of a VN is expressed as the amount of its nodes: $n = |N^V|$. Each pair of virtual nodes is randomly connected with probability 0.5. The CPU resources at nodes and the link data rates follow a uniform distribution from 0 to β units. For example, a β of 50 means that the CPU or data rate constraint of a virtual node or link is set to maximal 50 units. Thus VNs with large β values are potentially harder to map than VNs with lower β values.

4.2 Evaluation Results

All results are evaluated at a confidence level of 95 percent. The confidence intervals are not shown in the figures because they got too small.

4.3 Determining proper ϵ and ω values.

Figures 4 and 5 depict the impact of ω on mapping quality and runtime of vnmFlib for VNs of different size and β values. One can see that the runtime converges fast and the R/C-Ratio reaches its maximum around $\omega = n + 10$. Since our primary interest is mapping quality and overestimating does not increase the runtime too much we set $\omega = 4n$ for the rest of our experiments.

Figures 6 and 7 depict the impact of ϵ on mapping quality and runtime. As in case of the ω values the runtimes converge fast and thus an overestimation is not critical in this context. But overestimating of ϵ can decrease the R/C-Ratio as can be seen in Figure 6. The R/C-Ratios for the $n = 10$ VNs decline from over 0.65 for $\epsilon = 1$ to around 0.5 for $\epsilon = 2$ but the effect weakens for the $n = 20$ and $n = 40$ networks and dissapears for the $n = 60$ and $n = 80$ networks. Thus we chose two approaches for setting the ϵ value: A simple approach with constant $\epsilon = 10$ and a more advanced approach which tries to find a proper ϵ in the interval $[1, 10]$. The advanced vnmFlib starts with $\epsilon = 1$. If it cannot find a valid VNM ϵ is increased by one and the algorithm continues with the mapping process. The algorithm stops either if it finds a valid VNM or if $\epsilon > 10$.

4.4 Single VNRs.

We next compare runtimes and R/C-Ratios of simple vnmFlib, advanced vnmFlib and the two stage approach (2stage) of Ref. [15] for virtual network requests (VNRs) of various size and various β values. For the 2stage algorithm we used the implementation of Ref. [12]. Each VNR is mapped separately onto a PN such that the total resources of the PN are available to this single VNR.

Figures 8 and 9 depict runtimes for VNRs of size $n = \{10, 20, 30, 40\}$, $\beta = \{30, 90\}$ and the corresponding R/C-Ratios. For the smaller $\beta = 30$ value (solid lines) the runtimes and R/C-Ratios of 2stage (crosses) and simple vnmFlib (squares) are nearly equal while the advanced vnmFlib approach (triangles) needs slightly more time but produces better R/C-Ratios especially for VNs of size $n = 10$. For bigger β values (dashed lines) the runtime of 2stage increases strongly while the R/C-ratio decreases. This effect strengthens with growing network size. To map a VN of 40 nodes and $\beta = 90$ the 2stage approach takes nearly 11 seconds and produces a R/C-Ratio of 0.15. The vnmFlib algorithm reacts much more stable on increasing β values and VN size (note that the runtimes of simple vnmFlib for $\beta = 30$ and $\beta = 90$ overlap) and maps the same VNRs in about 1 second with a R/C-Ratio of 0.3. The advanced vnmFlib reaches a R/C-Ratio of 0.45 in less than 2 seconds. For smaller VNs the R/C-Ratio of the advanced vnmFlib algorithm is nearly twice as good compared to the R/C-Ratios of simple vnmFlib and 2stage. Note that for $n = 10$ and $\beta = 30$ the advanced vnmFlib algorithm achieves a nearly optimal R/C-Ratio.

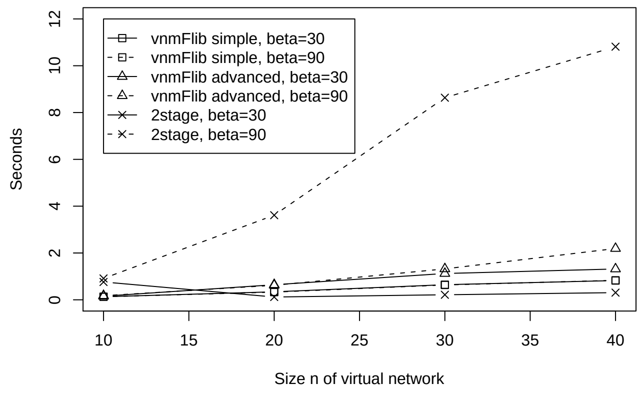

Figure 8: Runtimes of vnmFlib and 2stage algorithms for VNs of different size n and β values.

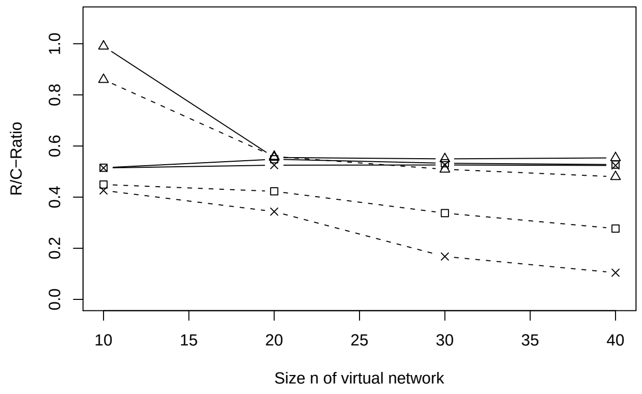

Figure 9: R/C-Ratios of vnmFlib and 2stage algorithms for VNs of different size n and β values.

4.5 Multiple VNRs and Path Splitting

To compare the performance of our algorithm with 2stage for dynamically arriving VNRs and mapping of multiple VNRs onto the same PN at the same time we implemented the environment of Ref. [15] as depicted in Figure 12.

The time axis is divided into a sequence of time windows of equal size $\triangle t$. At each time t the vnmFlib algorithm tries to map all virtual networks requests r_i with arrival times $a_i \leq t + \triangle t$. All requests with $a_i + l_i > t$ are deleted and there resources on the PN released.

Virtual Network Requests. In a VNR the number of nodes is randomly determined by a uniform distribution between 20 and 40 following similar setups to previous work [17, 15]. Each pair of virtual nodes is randomly connected with probability 0.5. The arrivals of the VNRs are modeled by a Poisson process with mean five requests per time window. The duration of the requests follows an exponential distribution with 10 time windows on average.

Note that we used the same experiment setup as in Ref.

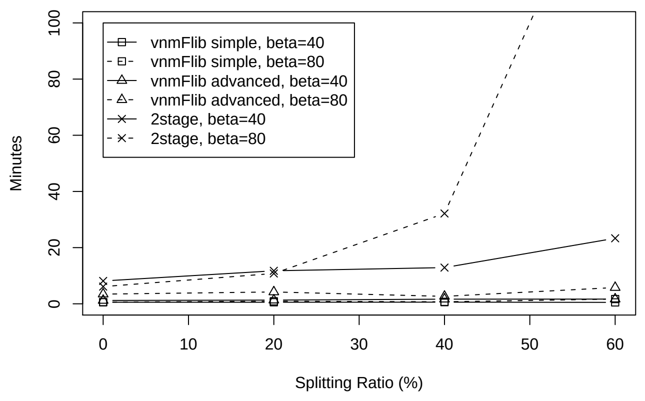

Figure 10: Runtime in relation to Splitting Ratio for vnmFlib and 2stage

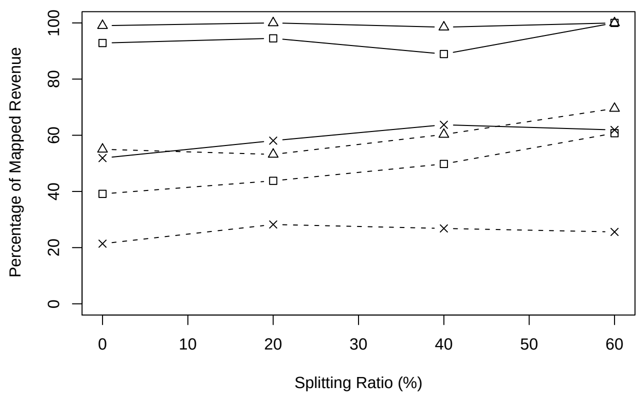

Figure 11: Percentage of total revenue in relation to Splitting Ratio for vnmFlib and 2stage

[15] but increased VN size. In Ref. [15] the sizes of the VNs vary between 2 and 20 nodes. We evaluated VN sizes between 20 and 40 nodes which seems to be a more realistic VN size. For the setup of the 2stage algorithm we had to adjust two parameters:

1. T_{try} which is the number of rounds in node remapping the algorithm does. Since our experiments showed that higher values do not improve the output quality significantly but increase the runtime we chose to set $T_{try} = 0$. These results are consistent with Ref. [15].

2. Allow path migration or not. Since we are primarily interested in mapping quality and as shown by Ref. [15] path migration improves mapping quality we allowed path migration.

Path Splitting. So far we have only considered the mapping of virtual links onto physical links or paths. By path splitting we mean that a virtual link can also be split up and mapped onto a flow of sufficient data rate. For a detailed discussion of path splitting see Ref. [15].

Figures 10 and 11 show the runtimes and percentage of mapped revenue for different β-values and splitting ratios. Splitting ratio denotes the percentage of VNRs that allow path splitting. By percentage of mapped revenue we mean the ratio of the total revenue of all VNRs to the revenue of all successfully mapped VNRs.

Mapping multiple VNRs onto the same PN at the same time is a challenging task for a VNM algorithm. Here the vnmFlib algorithm benefits from the high R/C-Ratios it achieves for the single mappings. Note that a high R/C-Ratio means that the physical resources consumed by a VNM are small. Thus if a VNM algorithm achieves higher R/C-Ratios he can map more VNRs onto the same PN at the same time. This results in fewer rejected VNRs and the algorithm obtains a higher revenue. This interrelation is reflected by Figure 11.

For $\beta = 40$ (dashed lines) the simple vnmFlib approach (squares) achieves around 20 percent more revenue for all splitting ratios than the 2stage algorithm (crosses). The advanced vnmFlib approach (triangles) is even better and achieves at least 35 percent more revenue. For $\beta = 80$ (solid lines) simple vnmFlib performs around 40 percent and advanced vnmFlib around 50 percent better than 2stage. This is due to the better R/C-Ratios of the VNMs produced by vnmFlib. As a consequence the VNMs of the vnmFlib algorithm need less resources of the PN than the VNMs produced by 2stage. Thus the vnmFlib algorithm can map more VNRs onto the same PN at the same time and achieves a higher revenue.

In addition to the higher revenue the vnmFlib algorithm is also faster than 2stage. For the $\beta = 40$ VNRs and 0 percent splitting ratio it takes about 8 minutes to map all VNRs with 2stage. This is due to the costly multi commodity flow computations which take place in the link mapping stage of 2stage. The simple vnmFlib can map all requests in 57 seconds and the advanced approach needs 90 seconds. This effect strengthens with increasing splitting ratio. For $\beta = 80$ VNRs and a splitting ratio of 60 percent 2stage needs about 167 minutes to map all VNRs while simple vnmFlib needs 2 minutes and the advanced approach 6 minutes.

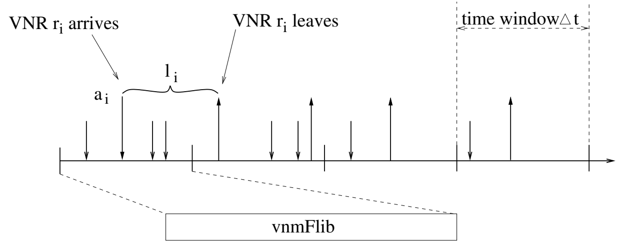

Figure 12: Evaluation environment for dynamicly arriving VNRs

5. CONCLUSION

In this paper we presented a virtual network mapping algorithm based on subgraph isomorphism detection which is able to handle multiple capacity constraints and dynamicly arriving online requests. We implemented a prototype and compared its performance with the two stage algorithm presented in Ref. [15].

Our evaluation results show that our subgraph-isomorphism based approach produces better mappings in less time than the two stage method. In particular for large networks with high capacity consumption our method performed better.

6. REFERENCES

[1] A. Bavier, N. Feamster, M. H., L. Peterson, and J. Rexford. In VINI Veritas: Realistic and Controlled Network Experimentation. In *Proc. of SIGCOMM*, pages 3–14, 2006.

[2] L. P. Cordella, P. Foggia, C. Sansone, and M. Vento. An Improved Algorithm for Matching Large Graphs. *3rd IAPR-TC15 Workshop on Graph-based Representations in Pattern Recognition*, pages 149–159, 2001.

[3] L.P. Cordella, P. Foggia, C. Sansone, and M. Vento. Performance Evaluation of the VF Graph Matching Algorithm. *Image Analysis and Processing, International Conference on*, 0:1172, 1999.

[4] Emulab – Network Emulation Testbed. http://www.emulab.net/.

[5] J. Fan and M. H. Ammar. Dynamic Topology Configuration in Service Overlay Networks: A Study of Reconfiguration Policies. In *Proc. of IEEE INFOCOMâĂŹ06*, 2006.

[6] N. Feamster, L. Gao, and J. Rexford. How to Lease the Internet in Your Spare Time. *SIGCOMM Comput. Commun. Rev.*, 37(1):61–64, 2007.

[7] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, 1979.

[8] GENI. http://www.geni.net.

[9] J. Lu and J. Turner. Efficient Mapping of Virtual Networks onto a Shared Substrate. Technical report, Washington University in St. Louis, 2006.

[10] Planetlab. https://www.planet-lab.org.

[11] R. Ricci, C. Alfeld, and J. Lepreau. A Solver for the Network Testbed Mapping Problem. *Computer Communications Review, 33(2)*, April 2003.

[12] Virtual Network Embedding Simulator. http://www.cs.princeton.edu/ minlanyu/embed.tar.gz.

[13] W. Szeto, Y. Iraqi, and R. Boutaba. A Multi-Commodity Flow Based Approach to Virtual Network Resource Allocation. *In: Proc. IEEE Global Telecommunications Conference (GLOBECOM'03). San Francisco, CA (USA).*, 2003.

[14] J. Turner and D. Taylor. Diversifying the Internet. In *Proc. of the Global Telecommunications Conference*, pages 755–760, 2004.

[15] M. Yu, Y. Yi, J. Rexford, and M. Chiang. Rethinking Virtual Network Embedding: Substrate Support for Path Splitting and Migration. *SIGCOMM Comput. Commun. Rev.*, 38(2):17–29, 2008.

[16] E. W. Zegura, K. L. Calvert, and S. Bhattacharjee. How to Model an Internetwork. In *INFOCOM (2)*, pages 594–602, 1996.

[17] Y. Zhu and M. Ammar. Algorithms for Assigning Substrate Network Resources to Virtual Network Components. *In Proc. of IEEE INFOCOMM06*, 2006.