A reactive resource defragmentation method for virtual links mapping in software-defined networks

A. F. Simo Tegueu*†, Slim Abdellatif*†, Thierry Villemur *‡, and Pascal Berthou*§
*CNRS, LAAS, 7 Avenue du colonel Roche, F-31400 Toulouse, France
†Univ de Toulouse, INSA, LAAS, F-31400 Toulouse, France
‡Univ de Toulouse, UT2J, LAAS, F-31100 Toulouse, France
§Univ de Toulouse, UPS, LAAS, F-31400 Toulouse, France

***Abstract*—Assigning network resources to Virtual Links (VLs) efficiently and on-demand is a challenging problem for any network virtualization solution. Known as the Virtual Link Mapping (VLM) problem, its objective is to compute the appropriate network paths with the required network resources that meet the quality of service expectations of arriving VLs while spreading the load over all nodes to maximise the admissibility of forthcoming VLs. Despite the efficiency of existing VL mapping algorithms, when resources are allocated and released over time due to the arrivals and departures of VLs, the network inevitably drift into a fragmented state (with nodes with very different loads) often causing a VLs request rejection that could have been avoided with a different resource allocation. In practice, defragmentation algorithms are used in complement to VL mapping algorithms to proactively or reactively (on the event of a VLs request refusal) trigger some VLs reallocation (or migration). In this paper, we propose an Integer-Linear program (ILP) based reactive defragmentation and VLs mapping algorithm for an SDN/OpenFlow network. In addition to selecting the VLs that should be migrated to reduce network defragmentation, our algorithm also computes the paths (and the associated resources) that support the previously rejected VLs. Our solution was evaluated on a real network topology and the experiments showed that our proposal outperforms existing approaches from the literature by about 12% in terms of acceptance rate with a gain on migration costs around 40%.**

I. Introduction

Network virtualization (NV) enables the co-existence of multiple concurrent VNs over the same SN in an independent and isolated way. It relies on algorithms commonly known as Virtual Network Embedding (VNE) algorithms to efficiently compute substrate resources for each hosted VNs. In this problem, a VN is a logical network with some of its elements (nodes and links) being virtual. A virtual node is an abstraction of a network device that is often hosted on a single physical node. The resources allocated to a virtual network device are as diverse such as CPU, volatile memory, network interfaces, storage, switching, etc. Similarly, a VL is an abstraction of a network link that is instantiated on one or multiple physical paths. It consumes transmission resources (i.e. physical links bandwidth) as well as switching resources at each traversed physical nodes. On-demand creation and adjustment of a VN are non-trivial. They involve numerous configuration operations witch are needed to instantiate virtual nodes and deploy their connecting VLs. Realizing these operations at hand is a time-consuming task as well as error prone.

The emerging SDN paradigm has been recognized as a powerful technology to overcome these problems, by enabling dynamic and automated configurations for a fast reliable and scalable deployment. While SDN advantages have been already highlighted, its use has also introduced some new constraints, namely, the limited capacity of forwarding tables, which is actually around a few thousands of entries [1] [2] in commodity SDN-compliant devices. In an SDN based virtualized environment, these switching resources are not only requested by virtual nodes, but they are also required to embed VLs. In fact, a number of flow rules have to be installed on auxiliary nodes i.e., nodes that are not part of the VN request, but are involved to set up the physical paths that host the VLs. These flow rules are mainly installed in network devices flow tables but may also be installed in their group tables when embedding point-to-multipoint VLs or when enabling path splitting.

Independently from the efficiency and optimality of the VL mapping algorithms, when resources are allocated and released over time due to the arrivals and departures of VLs, the network inevitably drift into a fragmented state where nodes and links may have very different loads. Such network resource fragmentation favors VLs request rejection even though, in most situations, this rejection could have been avoided with a different resource allocation. This is why a defragmentation algorithm is usually used to complement VL mapping algorithms to proactively or reactively (on the event of a VLs request refusal) trigger some VLs reallocations (or migrations). Their objective is to evenly spread the load leading to a reduction of network resource fragmentation and, as a consequence, an improved admissibility for forth coming VLs requests. To the best of our knowledge, only the work in [2] proposes a defragmentation algorithm for SDN networks, to the extent that switching resources are explicitly considered in addition to the bandwidth of links. The algorithm relies on a proactive approach that selects and triggers some VLs migrations when the load of some links or nodes exceeds a pre-specified threshold. In this paper, we propose a reactive Integer-Linear Program (ILP) based defragmentation and mapping algorithm for SDN networks that is triggered when a VLs request is rejected. The algorithm jointly selects the most promising VL candidates to migrate and, when possible, the resource allocations related to the

rejected request.

The remainder of this paper is organized as follows. In Section II, we discuss existing SN re-allocation approaches. In Section III, we describe the problem under study and explain the proposed ILP, before presenting our algorithm of selection in Section IV. Section V details the performance evaluation and discusses the obtained results. Last, Section VI concludes this paper.

II. Related work

VL mapping on an SDN infrastructure has been investigated [3] and a handful solutions [4] [5] [6] have been proposed. In general, these latter don't consider the possibility to reconfigure already mapped VLs while mapping new requests. This would avoid resource fragmentation and hence improve the admissibility of upcoming requests. However, many research works address resource fragmentation [2] [7] [8] [9] [10] [11] [12] [13] and propose in complement to VNE algorithms, a strategy for SN resource re-optimization, increasing by the way the embedding performance.

In general, these strategies are decomposed into three successive phases.

1) Controlling reconfiguration: As it is not conceivable to apply reconfiguration at continuous time, the objective is to determine when to trigger the process and at which conditions reconfiguration will be effective in order to avoid intensive SN instability, bandwidth overhead and eventual service disruption of reconfigured VNs. We distinguish two categories of approaches according to the trigger mode: periodic [13] [12] and event-based. This last category decomposes into three subcategories: (1) reactively triggered on VN request rejection; (2) proactively [2] [7] triggered by events like VN departure and (3) hybrid which can be triggered when a VN request embedding fails or/and on a VN expiration. Some solutions [2] systematically reconfigure already mapped VNs while others [13][10][7] condition the reconfiguration by the level of congestion of substrate entities.

2) Selecting candidate virtual components: During this phase, relevant virtual components (virtual links and/or nodes) that are candidate for reconfiguration are selected. The number of candidates is limited in order to limit the computation cost and the reconfiguration duration. The selection depends on many factors like the contribution of the virtual component on the fragmented resources and the expected benefits in terms of fragmentation reduction of a potential migration.

3) Remapping: At this last stage, virtual components previously selected are reallocated. We distinguish three dimensions of classification of remapping methods. The first one includes (1) integrated approaches [9] [8] that execute selecting and remapping in a coordinated manner at the same time with the aim of finding an optimal remapping solution while minimizing service disruption; and (2) separated approaches where selecting and remapping are performed independently. This class includes interrelated approaches [14] requiring specific information from the initial embedding strategy, and portable approaches that are able to operate with any initial embedding strategy. Finally, (3) the last dimension allows to distinguish the approaches that are restrained to reallocate virtual components individually, from those that treat many components simultaneously for a further efficient resource utilization.

TABLE I
Classification of SN resource re-optimization approaches

Refs.	Re-opt. mode	Triggering conditions	Component to migrate	Considered resources	Remapping method
[2]	Proactive	N/A	Virtual link	Flow table Bandwidth	Separated Portable Individual
[7]	Proactive	Link congestion	Virtual node and its attached VLs	Bandwidth	Separated Portable Individual
[8]	Reactive	N/A	Virtual node and link	CPU Bandwidth	Integrated Portable Simultaneous
[9]	Reactive	N/A	Virtual node and link	CPU Bandwidth	Integrated Portable Simultaneous
[11]	Reactive	N/A	Virtual node and link	CPU Bandwidth	Separated Interrelated Simultaneous
[10]	Hybrid	Link congestion	Virtual node and its attached VLs	Bandwidth	Separated Portable Individual
[12]	Periodic	N/A	Virtual link	Bandwidth	Integrate Portable Simultaneous
[13]	Periodic	Link and node congestion	Virtual topology	CPU Bandwidth	Separated Portable Individual
Our approach	**Reactive**	**N/A**	**Virtual link**	**Flow table Bandwidth**	**Integrated Portable Simultaneous**

Table I summarizes existing works and classifies the method that we are proposing in this paper.

III. Proposed algorithm

Algorithm 1 outlines the pseudo-code of the general behaviour of our proposed algorithm. As explained above, it is activated upon a VLs request rejection and then acts as a defragmentation algorithm by selecting a set of VLs to migrate (denoted as ζ and referred as "candidate VLs") and recomputing their allocations after including the previously rejected VLs request that caused its activation. The first step (line 5) selects the candidate VLs, followed by the release of their assigned resources (line 6). The selection is based on a heuristic algorithm presented in the next section. At the next step (line 7), selected VLs are simultaneously remapped with those composing the rejected request. Finally, if the VN is successfully mapped, then the new allocations are committed, else, the original allocations assigned to candidate VLs will be reconsidered and the VN request is definitely rejected. It is worth noting that in this latter scenario, no VLs migration is applied. In fact, such a situation is typically preceded by VLs requests that were successfully remapped with the help of VLs migrations. From our observations, if there is a definitive request rejection, the network is in an overload situation where a defragmentation strategy is of no help.

The remainder of this section presents the method used to compute the resource allocations of selected and initially rejected VLs (line 7).

Algorithm 1: Reactive VLM considering migration

Input : $G(V,E)$; K set of incoming VLs request; X set of currently-mapped VLs; $F_k\ \forall k \in X$; θ; N_{max}

1 **begin**
2 Initialize $Success \longleftarrow false; \zeta \longleftarrow \emptyset$
3 $Success \longleftarrow$ `Allocate`$(G(V,E), K)$
4 **if** *(*$Success = false$*)* **then**
5 $\zeta \longleftarrow$ `Select VLs`$(G(V,E), X, F_k, \theta, N_{max})$
6 Revert currently assigned allocations to ζ
7 $Success \longleftarrow$ `Solve ILP`$(G(V,E), K, \zeta)$
8 **if** *(Success)* **then**
9 Commit new assigned allocations to ζ
10 **else** Rollback currently assigned allocations to ζ
11

A. Physical Network Model

The physical network is modeled by a bidirectional graph $G = (V, E)$ where V ($|V|$) is the set of physical nodes (SDN switches) and E ($|E|, E \subseteq V \times V$) the set of physical links which operate in full-duplex mode. To each node $i \in V$ is associated a switching capacity U_i which is the maximum number of entries (i.e. size limit) of its flow table. The current size of node i flow table is denoted by U'_i. Each link $(i,j), i,j \in V$ is weighted by its bandwidth B_{ij}. Links are assumed to exhibit the same characteristics in both directions, i.e. $B_{ij} = B_{ji}$. The bandwidth that is currently assigned at link (i,j) by already admitted VLs is denoted by B'_{ij}.

B. Virtual network request model

The incoming VN request consists of a set of K VLs. Each VL k is characterised by:

- a source node $s_k \in V$, and a set of destination nodes $T_k \subseteq V \backslash \{s_k\}$ (when $|T_k| = 1$, the VL is point-to-point, otherwise it is point-to-multipoint);
- a bandwidth requirement of $b_k \in \mathbb{N}^*$

C. Initial allocations of VLs

The initial mapping of each candidate VL $k \in \zeta$ is a substrate path, denoted as F_k, which consists of a set of physical links and nodes. $F_k(i,j)$ refers to the amount of bandwidth used on link $(i,j) \in F_k$ by the VL k. Likewise, we denote by $L_k(i)$ and $N_k(i)$ respectively the number of entries that are allocated at node i flow table and group table to support VL k. These quantities are used during the resolution process to revert/dis-embed (Algorithm 1 : line 6) the allocations of these VLs. For each VL $k \in \zeta$, we use the binary $G_k(i,j)$ to indicate if some resources are assigned to k at the physical link (i,j).

D. Resource-related assignment variables

The output of our assignment problem is the set of routes (with the bandwidth allocations at each supporting physical link and the number of flow and group table entries at each traversed node) that support each of the VLs that compose the request as well as the VLs re-allocated. It is worth noting that since VLs may be point-to-multipoint, it is likely that resource allocations will be mutualised close to the source and as we get closer to destinations, they will tend to be more and more dedicated to specific destinations. As a consequence, basic assignment variables are related to a specific destination of a VL. In our model, we distinguish the following variables:

- $f_k^t(i,j)$ is an integer variable that represents the bandwidth allocated at link (i,j) to the packets of VL k that are flowing from the origin node s_k to a destination node t. More generally, $f_k(i,j)$ refers to the amount of bandwidth used on link (i,j) by the VL k, whatever the destination. It is set to the maximum of $f_k^t(i,j)$ for all $k \in (K \cup \zeta)$.
- $l_k(i)$ is an binary variable that indicates the switching resources consumed by VL k at node i. It is expressed as the number of entries that are installed in node i flow table to support VL k with the assumption that all entries consume the same amount of resources regardless of the complexity of the *match* operation and the related *instructions* to perform. In this work, we assume that each VL consumes 1 flow table entry at each traversed node. A node is traversed by a VL if at less one of its adjacent physical link supports VL. Formally:

$$\forall k \in K \cup \zeta, \forall i \in V, \forall (i,j) \in E : g_k(i,j) \leq l_k(i) \quad (1)$$

$$\forall k \in (K \cup \zeta), \forall i \in V : l_k(i) \leq \sum_{\substack{j \in V \\ (i,j) \in E}} g_k(i,j) \quad (2)$$

  where $g_k(i,j)$ is an intermediate binary variable that indicates if some bandwidth from link (i,j) is assigned to VL k or not. It is derived from another set of more focused intermediate variables $g_k^t(i,j)$ that reflects whether the flow of packets of VL k destined to t is supported by the physical link (i,j) (i.e. $g_k^t(i,j) = 0$ if $f_k^t(i,j) = 0$ and $g_k^t(i,j) = 1$ otherwise).
- $n_k(i)$ is a binary variable indicating if a group table entry is assigned to VL k at node i. A group table entry is consumed by a VL at a node, if it is split (for multipath) or duplicated (for multicast). In other words, if at less two of its adjacent physical links support the VL. Formally (equations 3 and 4):
  $\forall k \in (K \cup \zeta), \forall i \in V, \forall (i,j) \in E :$

$$n_k(i) \leq \left(\sum_{(i,j) \in E} g_k(i,j) \right) - g_k(i,j) \quad (3)$$

  $\forall k \in (K \cup \zeta), \forall i \in V, \forall (i,j_1) \neq (i,j_2) \in E :$

$$g_k(i,j_1) + g_k(i,j_2) - 1 \leq n_k(i) \quad (4)$$

- f_{max} which refers to maximum link utilization (when considering all network links) after request acceptance.
- u_{max} which similarly refers to maximum flow table utilization (when considering all network nodes) after request acceptance.

- z_k is a binary variable that indicates if the VL $k \in \zeta$ has been reallocated or not, regardless the complexity of the configurations to apply for achieving path migration.

E. Problem Constraints

The constraints on bandwidth allocations are described hereafter in equations 5 to 11. Equation 5 reflects the linearisation of the Max operator applied to variables $f_k^t(i,j)$ to get $f_k(i,j)$. Equations 6 and 7 have a similar purpose and focus respectively on f_{max} and u_{max} which are minimized by the objective function (as explained below).

$$\forall k \in K \cup \zeta, \forall (i,j) \in E, \forall t \in T_k : f_k^t(i,j) \leq f_k(i,j) \tag{5}$$

$$\forall (i,j) \in E : \frac{1}{B_{ij}} * \left(B'_{ij} + \sum_{k \in K \cup \zeta} f_k(i,j) \right) \leq f_{max} \tag{6}$$

$$\forall i \in V : \frac{1}{U_i} * \left(U'_i + \sum_{k \in K \cup \zeta} l_k(i) \right) \leq u_{max} \tag{7}$$

Equation 8 ensures that the bandwidth assigned to each VL k at link (i,j) does not exceed the remaining bandwidth. Equation 9 is the usual flow conservation constraints.

$$\forall (i,j) \in E : \sum_{k \in K \cup \zeta} f_k(i,j) \leq B_{ij} - B'_{ij} \tag{8}$$

$$\forall k \in K \cup \zeta, \forall t \in T_k, \forall i \in V : \sum_{j \in \Gamma(i)} (f_k^t(i,j) - f_k^t(j,i)) = \begin{cases} b_k & if\ i = s_k \\ -b_k & if\ i = t \\ 0 & else \end{cases} \tag{9}$$

Equation 10 is a channeling constraint between integer and binary variables: $f_k(i,j)$ and $g_k(i,j)$. It also constrains the VL k's bandwidth assignment at a physical link to the requested bandwidth b_k. Equation 11 constrains the bandwidth that is assigned to the flow of packets destined to a specific VL's end-point (or destination) within a range of values, in addition to establishing a channeling constraints between binary and integer variables. The inequalty on the right side ensures that the bandwidth requirement of the VL is never exceeded. The inequalty on the left side directs path-spliting and avoids the multiplication of splits with low bandwidth allocations. Indeed, if active, path-splitting is feasible only if the bandwidth allocated to the splits respects a minimum threshold b_k^{min}. In practice, b_k^{min} is a ratio of b_k, $b_k^{min} = PS_{ratio} * b_k$ with $PS_{ratio} \in [0,1]$ (then, $PS_{ratio} \leq 0.5$ when the path-splitting is allowed, and $PS_{ratio} = 1.0$ when it is forbidden).

$$\forall k \in K \cup \zeta, \forall (i,j) \in E : g_k(i,j) \leq f_k(i,j)\ and\ f_k(i,j) \leq b_k * g_k(i,j) \tag{10}$$

$$\forall k \in K \cup \zeta, \forall (i,j) \in E : b_k^{min} * g_k^t(i,j) \leq f_k^t(i,j)\ and\ f_k^t(i,j) \leq b_k * g_k^t(i,j) \tag{11}$$

The constraints related to switching resources allocation is given by inequalities 12 and 13. They simply ensure that the total number of flow and group table entries assigned to candidate VLs and to the VLs composing the request, don't exceed available nodes' flow and group tables entries.

$$\forall i \in V : \sum_{k \in K \cup \zeta} l_k(i) \leq U_i - U'_i \tag{12}$$

$$\forall i \in V : \sum_{k \in K \cup \zeta} n_k(i) \leq N_i - N'_i \tag{13}$$

All the above cited constraints guarantee the simultaneous embedding of the rejected VN request and the candidate VLs. However, as our formulation aims to be sensitive to the migration cost including service disruption experienced by reconfigured VLs and also the induced overhead, for each of candidate VL $k \in \zeta$, we use binary variables z_k that indicate if the VL k is reallocated or not. They are used in the objective function to minimize the migration cost. z_k is defined as follows. A VL $k \in \zeta$ is changed at a physical link (i,j) if: (1) there was previously supported ($G_k(i,j) = 1$), but it is no longer the case ($g_k(i,j) = 0$), or (2) if it is newly allocated at this physical link ($G_k(i,j) = 0$ and $g_k(i,j) = 1$). These changes can be detected by this logical expression: $G_k(i,j) * \overline{g_k(i,j)} + \overline{G_k(i,j)} * g_k(i,j)$. To detect the total number of changes along the path, we have introduced an intermediate integer expression denoted y_k, defined as:

$$y_k = \sum_{(i,j) \in E} y_k(i,j)$$

Where $y_k(i,j)$ is a linear expression defined as follows: $y_k(i,j) = G_k(i,j) * (1 - g_k(i,j)) + (1 - G_k(i,j)) * g_k(i,j)$ which is derived from the previous logical expression. In this work, we also assume that any number of changes in the original path of a VL is counted as only one reconfiguration regardless of the number of physical links that are changed. Then, the variable z_k can be derived from y_k like this:

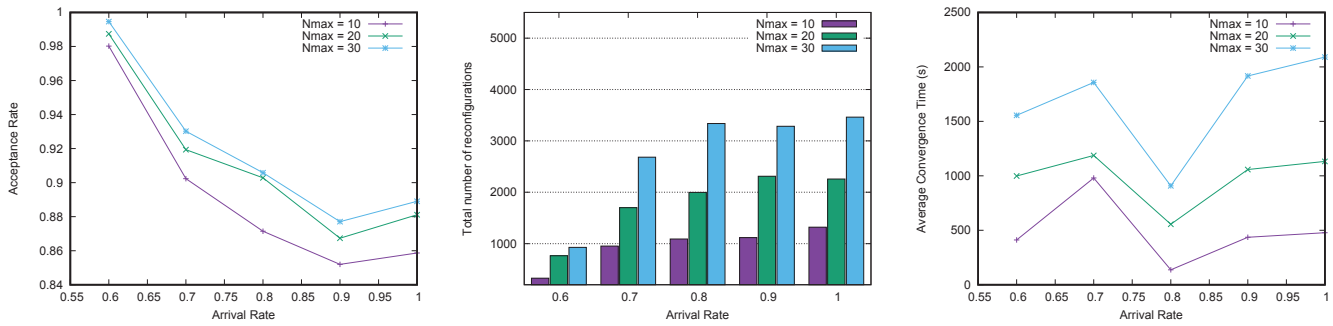
$$\forall k \in \zeta : z_k = \begin{cases} 0, & if\ y_k = 0 \\ 1, & if\ y_k \geq 1 \end{cases}$$

As z_k is a binary variable, the above equality can be linearly represented as follows:

$$\frac{y_k}{2 * |E|} \leq z_k \leq y_k \tag{14}$$

F. Objective function

The objective function aims at minimizing link and node resource consumption but also at distributing the consumed resources among nodes and links in order to reduce the creation of bottlenecks while minimizing migration cost. Therefore, the admissibility of forthcoming requests are improved, at the same time, avoiding to frequently trigger reconfigurations. As shown in expression 15, the objective function consists of five components, each weighted with a parameter that controls the impact of the component on the resolution process. The first two concern bandwidth allocations and the next two are their analogue for flow table entries allocations. Finally, the



(a) Impact on Acceptance Ratio (b) Impact on Migration Cost (c) Impact on Solving Time

Fig. 1. Impact of parameter N_{max}

last component concerns migration cost in terms of the total number of reconfigured VLs.

$$\begin{aligned}\text{minimize } & \alpha_1 * \frac{1}{|E|} * \sum_{(i,j)\in E} \left(\frac{1}{B_{ij}} * \left(B'_{ij} + \sum_{k\in K\cup\zeta} f_k(i,j) \right) \right) \\ & \alpha_2 * f_{max} + \\ & \beta_1 * \frac{1}{|V|} * \sum_{i\in V} \left(\frac{1}{U_i} * \left(U'_i + \sum_{k\in K\cup\zeta} l_k(i) \right) \right) + \\ & \beta_2 * u_{max} + \rho * \sum_{k\in\zeta} z_k \end{aligned} \quad (15)$$

IV. Heuristic algorithm for VLs pre-selection

Even though all current VLs can be re-allocated, our experiments show that only a portion of existing VLs is profitably reallocated. In this section, we introduce a simple heuristic algorithm to pre-select the VLs that will be include in the process of reallocation. This reduces the size of the ILP problem and hence makes it tractable for large networks.

Unlike some approaches proposing to consider all VLs from the same VN that are mapped on θ-congested entities (A SN node i and link (i,j) are considered to be θ-congested if $\frac{U'_i}{U_i} \geq \theta$ and $\frac{B'_{ij}}{B_{ij}} \geq \theta$ respectively), we propose to limit the number of VLs to migrate to N_{max}. Since our proposal operates with no information on future requests, the principle relies on the notion that we called "Minimum Spanning Set (MSS)", which is the subset of VLs with smallest cardinality, hosted by all θ-congested entities. To build MSS, the algorithm iterates over the set of VLs that are hosted by at less one θ-congested entity, to select at each iteration, among the most popular/impacting (in terms of number entities that it congests among those not yet spanned). The MSS is built when all θ-congested entities are covered. The VLs composing the MSS are selected, then a new MSS is computed until the total number of selected VLs reaches N_{max}.

V. Performance evaluation

We firstly introduce the simulation settings. Then, we show the impact of some parameters, before presenting a comparative analysis to existing solutions.

A. Simulation settings

1) Network model: We consider in this work a real network topology taken from the European Research Network GEANT with 41 network nodes and 60 links that connect the main European cities. We adopt the link capacities that are given by GEANT. Finally, we assume that a flow table size of 125 entries is dedicated at each node to the considered VLs resource allocations.

2) Load model: We assume that requests arrive following a Poisson distribution with an arrival rate λ that is varied on $\{0.6, 0.7, 0.8, 0.9, 1\}$ i.e. the average number of requests varies from 60 to 100 requests per 100 units of time (UT) respectively. The requests lifetime conforms to an exponential distribution with an average of 125UT. The number of VLs per request is set according to a discrete uniform distribution, using the values given in $[6, 12]$. The bandwidth requirement is uniformly distributed between 100 and 300Mbps.

3) Algorithms settings: We implemented our ILP formulation, using concert technologies C++ as the modeling layer and IBM CPLEX 12.6 as solver. The resolution time of the solver is set to a maximum of 3600 seconds. A gap of less than 5% to the optimal solution is considered satisfactory. The simulation horizon is fixed to 1500UT. The fixed Parameters $\alpha 1$, $\alpha 2$, $\beta 1$, $\beta 2$ and ρ of the objective function are set to 1, 100, 1, 300, 100 respectively. The parameters of the selection algorithm are set as follows: $\theta = 0.95$, N_{max} is varied over $\{10, 20, 30\}$. Our solution is compared to "SDN-VN" calibrated like in [2].

B. Evaluation results and analysis

1) Effects of N_{max}: We first examine the impact of N_{max} on the different performance metrics by analyzing the results illustrated in Figures 1a, 1b and 1c. Before that, let's introduce Figure 2, which shows the total number of triggers having resulted in a successfully mapping and the average number of reallocated VLs at each trigger as a function of the arrival rate. An important observation is that, even though only N_{max} over all running VLs are involved in the reallocation process, only a fraction by around 80% are migrated. In one side, this shows that not all VLs deserves being migrated. On the other hand, 80% is a satisfactory score revealing the

efficiency of pre-selection algorithm. From Figure 1c, we evidently observe that the average convergence time of our ILP considerably reduces when N_{max} decreases. Figure 1a shows the average acceptance ratio for different values of N_{max}. Obviously, the acceptance ratio increases with the number of VLs involved in the reallocation. The reason is that, reallocating more VLs gives more chance to unload over loaded substrate links and nodes, hence promoting the requests admittance. Unfortunately, the migration cost also grows as depicted in Figure 1b. It is clear that the migration cost and acceptance ratio are conflictual metrics leading to a compromise, which must be treated according to the use case.

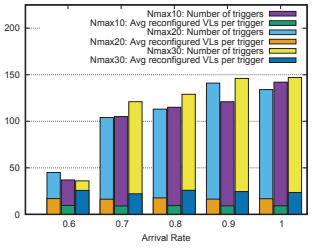
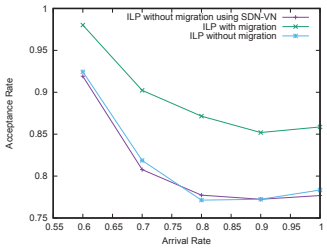
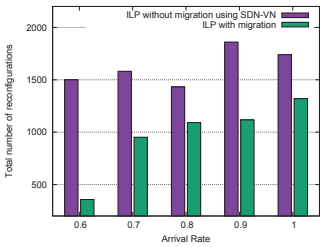


Fig. 2. Reconfigurations over triggers

2) Comparative analysis: The final experiments show the gain of our proposal, compared to ILP without migration and also with ILP without migration but operating with the proactive approach "SDN-VN". The results from Figure 3a show that reallocation improves the requests admissibility. It also shows that, our proposal achieves better acceptance rate than "SDN-VN", as well as a lower migration cost.



(a) Comparison on Acceptance Rate



(b) Comparison on Migration Cost

Fig. 3. Comparisons ($N_{max} = 10$)

VI. Conclusion

This paper has proposed a reactive defragmentation and remapping Integer-Linear Program method for an efficient online virtual links mapping in SDN networks. The proposed method supports point-to-point and also point-to-multipoint VLs each with an associated bandwidth requirement. Their remapping takes into account some of the specificities of SDN, namely the current limitation of switching resources (size of the flow and group tables). Another characteristic is that it jointly selects an appropriate set of VLs which will be migrated while mapping the newly rejected VLs request.

Our solution was evaluated on a real network topology and compared to the SDN-VN method. The simulations showed that our proposal improves the admissibility of VLs requests and outperforms SDN-VN by 12% in terms of acceptance rate with a gain on migration costs around 40%.

These preliminary results will be pursued. In our future work, experiments will be extended in order to assess the influence of parameter θ. Also, Point-to-Multipoint VLs will be considered in the performance evaluations.

Acknowledgment

This work was partially funded by the French National Research Agency (ANR) and the French Defense Agency (DGA) under the project ANR DGA ADN (ANR-13-ASTR-0024) and by European Unions Horizon 2020 research and innovation programme under the ENDEAVOUR project (grant agreement 644960).

References

[1] B. A. A. Nunes, M. Mendonca, X. N. Nguyen, K. Obraczka, and T. Turletti, "A survey of software-defined networking: Past, present, and future of programmable networks," *IEEE Communications Surveys Tutorials*, vol. 16, no. 3, pp. 1617–1634, Third 2014.

[2] R. Mijumbi, J. Serrat, J. Rubio-Loyola, N. Bouten, F. De Turck, and S. Latr, "Dynamic resource management in sdn-based virtualized networks," in *10th International Conference on Network and Service Management (CNSM) and Workshop*, Nov 2014, pp. 412–417.

[3] M. Demirci and M. Ammar, "Design and analysis of techniques for mapping virtual networks to software-defined network substrates," *Computer Communications*, vol. 45, pp. 1 – 10, 2014.

[4] L. R. Bays, L. P. Gaspary, R. Ahmed, and R. Boutaba, "Virtual network embedding in software-defined networks," in *NOMS 2016 - 2016 IEEE/IFIP Network Operations and Management Symposium*, April 2016, pp. 10–18.

[5] F. S. Tegueu, S. Abdellatif, T. Villemur, P. Berthou, and T. Plesse, "Towards application driven networking," in *2016 IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN)*, June 2016, pp. 1–6.

[6] R. Trivisonno, I. Vaishnavi, R. Guerzoni, Z. Despotovic, A. Hecker, S. Beker, and D. Soldani, "Virtual links mapping in future sdn-enabled networks," in *Future Networks and Services (SDN4FNS), 2013 IEEE SDN for*, Nov 2013, pp. 1–5.

[7] B. Wanis, N. Samaan, and A. Karmouch, "Substrate network house cleaning via live virtual network migration," in *2013 IEEE International Conference on Communications (ICC)*, June 2013, pp. 2256–2261.

[8] T. P. N. and T.-G. A., "Reconfiguration of virtual network mapping considering service disruption," in *2013 IEEE International Conference on Communications (ICC)*, June 2013, pp. 3487–3492.

[9] P. N. Tran, L. Casucci, and A. Timm-Giel, "Optimal mapping of virtual networks considering reactive reconfiguration," in *2012 IEEE 1st International Conference on Cloud Networking (CLOUDNET)*, Nov 2012, pp. 35–40.

[10] I. Fajjari, N. Aitsaadi, G. Pujolle, and H. Zimmermann, "Vnr algorithm: A greedy approach for virtual networks reconfigurations," in *Global Telecommunications Conference (GLOBECOM 2011), 2011 IEEE*, Dec 2011, pp. 1–6.

[11] E. Estrada and D. J. Higham, "Network properties revealed through matrix functions," *SIAM Review*, vol. 52, no. 4, pp. 696–714, jan 2010.

[12] M. Yu, Y. Yi, J. Rexford, and M. Chiang, "Rethinking virtual network embedding: Substrate support for path splitting and migration," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 17–29, Mar. 2008.

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

[14] N. Farooq Butt, M. Chowdhury, and R. Boutaba, *Topology-Awareness and Reoptimization Mechanism for Virtual Network Embedding*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 27–39.