Network Defragmentation in Virtualized Data Centers

Oliver Michel
University of Colorado Boulder
oliver.michel@colorado.edu

Eric Keller
University of Colorado Boulder
eric.keller@colorado.edu

Fernando M. V. Ramos
LaSIGE, Faculdade de Ciências
Universidade de Lisboa, Portugal
fvramos@ciencias.ulisboa.pt

***Abstract*—Network virtualization is an extensively used approach to allow multiple tenants with different network architectures and services to coexist on a shared data center infrastructure. Core to its realization is the mapping (or embedding) of virtual networks onto the underlying substrate infrastructure. Existing approaches are not suitable for cloud environments as they lack its most fundamental requirement: elasticity. To address this issue, we introduce two new network primitives – *expand* and *contract* – which allow virtual networks to scale up and down. Mapping and scaling virtual networks over time, however, introduces fragmentation in the substrate network. This is akin to fragmentation in a file system where files are not laid out in contiguous physical blocks of the storage device. This problem impacts network performance and reliability for tenants and their applications. Instead of further improving embedding algorithms to tackle this problem, in this work, we present a yet unexplored approach: leveraging network migration techniques to *defragment* the network. We introduce network defragmentation as a new management primitive and propose algorithms to materialize it. We show through extensive simulations that our techniques significantly improve network performance while maintaining high utilization of the infrastructure, thus increasing provider revenue. On average, using defragmentation leads to 20% reduction in path length and utilization and cuts the number of very long paths (longer than half of the network diameter) between 52% and 62%. Moreover, it doubles the number of servers utilized by 50% or less as a result of consolidation.**

I. Introduction

Network virtualization is a widely used technique that enables a more dynamic infrastructure in the context of data center networks. Being a cornerstone of cloud computing, virtual networks (VN) should ideally provide an abstraction that makes tenant infrastructure more elastic by allowing them to scale up and down their networks depending on demand, by adding, removing, expanding, or contracting their virtual networks dynamically. To achieve this, it is necessary to map the virtual network requests onto the substrate infrastructure, a problem commonly known as Virtual Network Embedding (VNE).

The literature on this topic is rich [12], but most existing approaches are not suitable for a cloud environment as they do not fit the model of dynamic scaling, limiting the introduction of modern networking services [18] in the cloud. To address this issue, we introduce two network primitives to VNE – *expand* and *contract* – providing the necessary elasticity to this environment.

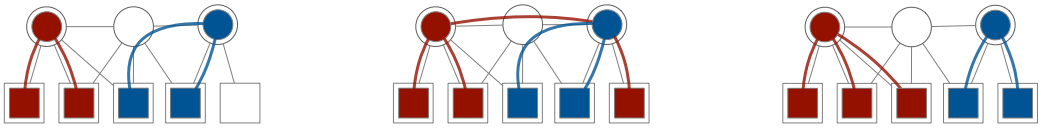
Performing the mapping and scaling operations frequently tends to introduce fragmentation in the underlying substrate. This is analogous to fragmentation in a file system where files are not laid out in contiguous physical blocks of the storage device. When allowing virtual networks to scale up or down (expand and contract) over time, we observe that the resource fragmentation effect becomes even more substantial.

When the physical network is fragmented, path lengths in the virtual networks increase as resources belonging to the same virtual network are mapped across distant regions of the data center. As a result, communication between adjacent nodes (in the virtual topology) needs to traverse additional switches, greatly increasing latency and bandwidth variability between virtual machines, and thus impacting job completion times and overall application performance [25]. In addition, physical resources increasingly become unusable with a decline of the virtual network acceptance rate, leading to a decrease in revenue for the infrastructure provider (InP), even though the physical network technically still has sufficient capacity to accommodate a request.

Instead of further optimizing VNE to address the fragmentation problem, we propose a different approach: leveraging network migration techniques to remap the substrate network with the goal of both improving substrate utilization and boosting network performance.

Virtual machine migration approaches have been proposed to optimize and consolidate machine placement, reducing fragmentation at the hypervisor-level, but these solutions generally do not take the network that interconnects the virtual machines into account. This is particularly relevant in the context of data center networks that are increasingly programmable, with Software Defined Networking (SDN) based control and reconfigurable data planes. In these modern data centers, a tenant can run their own network applications on virtual network devices connected through links with reserved, guaranteed properties. This motivates us to specifically target the domain of virtualized data center networks with abstractions for compute nodes (VMs), as well as networking infrastructure (programmable switches and links).

Toward the goal of mitigating the effects of resource fragmentation, we propose *network defragmentation* as a new network management primitive. The idea is to explicitly remap virtual networks when fragmentation reaches a certain threshold, in order to optimize network performance and resource usage, effectively de-fragmenting the network. This concept is depicted in Figure 1. To implement this primitive, we envision using modern virtual machine and network migration techniques, such as LIME [15], which are able to migrate entire virtual networks including virtual machines, routers, and links, with negligible downtime.



(a) before scaling (b) fragmented network (c) after defragmentation

Fig. 1: The network became fragmented after adding an additional node to the red network, after the blue network was embedded, resulting in an increased path length that is corrected using defragmentation.

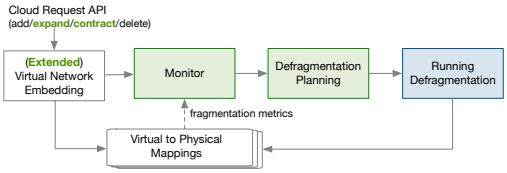


Fig. 2: System Architecture

Figure 2 illustrates the entire system, with the current state of network virtualization shown in white, our new extensions shown in green, and future work beyond the scope of this paper shown in blue. In this, a cloud API allows users to make requests which are then embedded, resulting in a mapping of the virtual networks. A monitor observes key metrics to decide when to trigger a defragmentation. Defragmentation planning would then decide which networks to migrate and where, resulting in a target mapping. The final step is to actually run the defragmentation, where the migration steps to get the network to this state must be planned and carried out.

To summarize, we make the following contributions:

1) We elaborate on new requirements for virtual network embedding in the cloud (namely, dynamic scaling of networks) and provide a solution that allows for expansion and contraction of virtual networks, matching the way cloud infrastructure is used in practice (Section III).
2) We identify challenges associated with scaling of virtual networks in the cloud and elaborate on metrics to quantify the resource fragmentation problem (Section IV).
3) We propose a new network management primitive – network defragmentation – and a set of defragmentation heuristics to materialize it, which we demonstrate to improve both network resource usage and network performance (Section V).
4) We evaluate our approach using extensive discrete event simulations and make our cloud simulation framework publicly available as open-source software[1]. We show that defragmentation on average leads to 20% reduction in path length and utilization, cuts the number of very long paths (longer than half of the network diameter) between 52% and 62%, and more than doubles the numbers of servers utilized by 50% or less through resource utilization consolidation.

II. Related Work

In this paper we propose network defragmentation as a means of overcoming the challenges that arise in both tenant network performance and provider efficiency due to the dynamic behavior of modern cloud infrastructures. While there has been a great deal of research in virtual network embedding toward parts of this objective, our work is the only to address a comprehensive solution, as summarized in Table I. We now summarize the most related work subdivided by the two main goals – suitability for cloud environments and network defragmentation.

A. Suitability for Cloud Environments

Cloud infrastructures have unique properties which make them a unique challenge for virtual network embedding. Here, we outline these key characteristics and their implications.

1) Substrate Topologies: For reasons of scalability, failure resiliency and to provide predictable bisection bandwidth, data center network topologies usually have the form of a multi-rooted tree (*e.g.,* fat tree, leaf-spine) or some sort of cube (*e.g.,* BCube). The majority of work in the area of virtual network embedding though is designed for and evaluated with mesh-like physical topologies, which are commonly found in wide-area networks [5], [6], [11], [14], [29]. While they may be able to handle topologies more commonly found in data centers, our evaluation with [6], a poweful VNE algorithm, showed that when using tree-like topologies the algorithm's performance (in terms of acceptance ratio) suffers greatly.

2) Virtual Topologies: Different cloud applications, like analytics frameworks or web application infrastructures, typically require different network topologies, such as multiple groups of VMs interconnected to each other via a central switch or a tiered layer 3 network layout. In addition, specific links in such a topology must provide certain QoS guarantees. Most existing solutions fall short in terms of flexibility for tenants by limiting the possible virtual network topologies [18]. For instance, [3], [13], [23], and [16] all only provide a limited set of topologies. This simplification allows them to use specialized (often bin-packing based) VNE approaches as opposed to providing a generalized VNE scheme. In some, *e.g.,* [16], it is not possible to define virtual topologies at all. Instead, all VMs are all part of a single broadcast domain which is logically equivalent to a star topology with a single virtual switch. [26] and [11] support generalized topologies, but are evaluated on random substrate networks, which simplifies the embedding as a random graph is mapped on a random graph. [28] and [22] are exceptions as they both work with arbitrary topologies and evaluate these on data center substrate topologies.

3) Network Control: Controlling the network means being able to specify custom forwarding logic, middlebox functionality, and traffic engineering. With more complicated workloads, control and QoS guarantees in the network will lead to better performance. With full SDN virtualization approaches available [8], [24], this capability is possible, and we believe cloud providers will eventually provide greater network control (a trend that has already

[1]available at: https://github.com/olivermichel/dcdefrag

TABLE I: Overview of Related Work

	Suitability for Cloud Environments				Network Defragmentation		
	DC Substrate Topo.	Virtual Topo.	Network Control	VN Scaling	Use Migration	Frag. Metrics	Defragmentation
SecondNet [16]	**yes**	limited	no	**yes**	**yes**	no	partially
VNE Evolving Net. [9]	no	**arbitrary**	no	no	**yes**	no	no
Reliable Virtual Infras. [26]	no	**arbitrary**	no	no	no	no	no
VNR Algorithm [11]	no	**arbitrary**	unclear	no	**yes**	**yes**	**yes**
Oktopus [3]	**yes**	limited	**yes**	no	no	no	no
VDC Embedding [22]	**yes**	**arbitrary**	**yes**	no	no	no	no
VDC Planner [28]	**yes**	**arbitrary**	**yes**	**yes**	**yes**	no	partially
Beyond the Stars [23]	**yes**	limited	partially	no	no	no	no
Kraken [13]	**yes**	limited	partially	**yes**	**yes**	no	no
DCDefrag (this work)	**yes**	**arbitrary**	**yes**	**yes**	**yes**	**yes**	**yes**

started with offerings like Elastic Load Balancing [2]). In order to support network control, the VNE algorithm must at least differentiate between different node types (*i.e.,* servers and switches). Many solutions however, only support a single logical virtual switch that is actually implemented through direct point to point tunnels between all VMs that need to communicate (*e.g.,* in [16]). [23] even removes the logical virtual switch as an optimization mechanism, and also directly connects all communication partners, which requires prior knowledge of the traffic matrix. With the majority of other work (*e.g.,* [13], [26], [28]) it is unclear how virtual switches can be connected to VMs, and whether the model actually supports network programmability or just basic layer 2 switching within a single broadcast domain.

4) VN Scaling: One of the main benefits of a virtualized compute and networking infrastructure (especially in a public cloud setting) is the ability to quickly and easily request resources as they become needed. This is opposed to buying hardware upfront and over-provisioning the data center in order to handle sudden increases in traffic, or even expected short-term changes in demand. As such, the network topology needs to dynamically adapt to the changing application needs. Most VNE work ignores the requirement to dynamically scale existing (already embedded) virtual networks. Exceptions to this are [16], [28], and [13]. While [9] targets *evolving networks*, the paper deals with evolving substrate networks and thus solves a different problem.

B. Network Defragmentation

Our solution is unique in that it not only solves a problem specifically suitable for cloud infrastructures, but it also addresses the fragmentation problem. We outline the required characteristics required towards this goal.

1) Use of Migration: Machine and network migration techniques, such as LIME [15] are able to migrate entire virtual networks including virtual machines, routers, and links, with negligible downtime. This technology can be tremendously useful for data center virtualization solutions as entire virtual networks can be moved and consolidated either during the VNE process (*i.e.,* to increase the chances of a successful embedding) or as part of an optimization mechanism to defragment the network as proposed in this paper. [13] uses virtual machine migration during the embedding process to improve the VN acceptance rate. [16] leverages migration to consolidate machine placement but uses a different network model. The closest solution in this aspect is [28], which periodically uses migration to optimize the network.

2) Fragmentation Metrics: Through simulations, we learned that performing mapping and scaling operations frequently tends to introduce fragmentation in the underlying substrate network (cf. Section IV). When allowing virtual networks to scale up and down (expand and contract) over time, we observe that the resource fragmentation effect becomes even more substantial. We define metrics for the physical network fragmentation problem and show that defragmentation can successfully optimize the network. The only related work that defines metrics for the network fragmentation problem and discusses causes for fragmentation is [11].

3) Defragmentation: While some VNE algorithms are able to limit the degree of fragmentation in the network when embedding a new network through load balancing mechanisms (*e.g.,* [22] and [6]), the problem becomes intractable when allowing networks to grow and shrink, as the embedding problem is significantly more constrained. Moreover, it is difficult to optimize for a global metric (fragmentation) in an online fashion without having knowledge about future requests. This is why we believe that a dedicated network defragmentation algorithm is a better approach toward reducing fragmentation in the network, improving tenant performance, as well as InP revenue. Network Defragmentation should be a network management primitive and, therefore, a requirement for cloud virtualization solutions. Most related to the network defragmentation primitive we introduce is the work that leverages VM migration to optimize data centers [10], [19]. Regarding leveraging network migration, Yu et.al. proposed path migration (re-mapping virtual links) to enable future virtual network embeddings to succeed [27]. [16] and [11] propose network optimization algorithms toward our above stated goals, but use entirely different network models which highly simplify the fragmentation problem. [28] periodically reruns the bin-packing based VNE algorithm to consolidate network resources in order to improve VN acceptance rates in the future. There is no study or quantification of fragmentation except for acceptance ratio, which makes it hard to evaluate the efficacy of the approach. Also, tenant network performance remains unaddressed. We go further and introduce a network-wide primitive for leveraging virtual network migration capabilities to optimize data centers for *utilization* and *performance*.

In summary, all related work only addresses a subset of the above described requirements and characteristics. To the best of our knowledge, our study and approach is the first toward a comprehensive cloud virtualization solution

that addresses the unique requirements in the cloud, as well as the consequences of prolonged cloud operation (*i.e.,* substrate fragmentation), and proposes a solution to this problem.

III. Dynamic VN Embedding for Data Centers

Given the shortcomings of existing VNE work to address the needs of modern data centers, here we present a new virtual network embedding solution for highly dynamic infrastructures. First, we define the general VNE problem. Then, we detail our network model which focuses on physical and virtual topologies of modern cloud systems. For this setting, we provide an extended problem definition. Second, we detail a recent VNE algorithm which we leveraged as it is best suited for these topologies. Third, we introduce two VNE primitives, expand and contract, along with their associated algorithms, which allow networks to match the dynamic behavior of cloud systems. Finally, we describe our implementation of these algorithms within a cloud simulator, which emulates cloud workloads using these new primitives.

A. VNE Problem Definition

The Virtual Network Embedding Problem can be formally defined as follows: Let $S = (V_S, E_S)$ be a substrate network consisting of substrate nodes V_S and substrate links (edges) E_S, and let $R_i = (V_{R_i}, E_{R_i})$ be the ith request for a new virtual network consisting of nodes V_{R_i} and links E_{R_i}. Each substrate resource has scalar capacity $c(s)$ for resource $s \in S$. Each virtual resource has scalar demand $d(r)$ for resource $r \in R$. Upon mapping resource r onto resource s, $d(r)$ is subtracted from the residual capacity $\Delta(s)$ of s. Δ of resource s at a given time is thus defined as $\Delta(s) = c(s) - \sum_{\forall v \to s} d(v)$, where $v \to s$ denotes that v has been mapped onto s. A virtual network embedding algorithm is defined by a function f that maps each $v \in V_R$ and each $e \in E_R$ onto one and only one $v \in V_S$ and $e \in E_S$, respectively: $f : v \to s$, s.t. $\Delta(s) \geq d(v) \geq 0$. f allows multiple virtual resources to be mapped on a single physical resource and can be defined differently for different resource types. It is important to note that node and link mapping must be carried out in a coordinated fashion in order to keep the exact original (virtual) topology.

B. Network Model

For the following discussion (and simulations), we assume that tenants want to exactly specify the network topology, and reserve resources on servers as well as on networking devices (switches or routers), which are then interconnected through reserved links with guaranteed bandwidth. For the physical network we adopt the model of Rabbani et.al. [21], which currently does not account for soft-switches on the hypervisor, instead every virtual switch is mapped onto a physical switch. That also means that every virtual link is mapped onto a physical link, even when two VMs which are connected to each other are located on the same physical server.

While our simulator and modified embedding scheme can handle arbitrary virtual and physical network topologies, for the experiments in this paper and our following evaluation, we leveraged a specific physical topology, as well as a mix of three different topologies for virtual networks.

1) Physical Network: Traditionally, data centers leveraged a core/aggregation/access layer network using the spanning tree protocol (STP) to disable redundant links avoiding forwarding loops. More recently, however, the leaf-spine network architecture has emerged and is becoming more popular than traditional data center topologies [4]. In such a network a set of spine switches is connected in a full mesh to a set of leaf switches which then connect to the individual servers.

2) Virtual Networks: For the virtual networks, while supporting arbitrary topologies, for our simulations we use three different topologies: The first two, Virtual Cluster (VC) and Virtual Oversubscribed Cluster (VOC) are taken from [3]. A VC is defined by a single switch that connects to N VMs in a star-topology with link bandwidth B. The VMs and the switch all have the same demand C. A virtual oversubscribed cluster consists of multiple virtual clusters that are connected to a central switch over links that are oversubscribed (in terms of bandwidth availability) by factor O. In this topology, N VM's in M groups of size S are connected to their group switch with bandwidth B. The group switches are connected to the main switch with bandwidth $B \times S/O$. The third topology is a three-tier topology commonly used in web workloads, where a layer of α load balancers is connected to a layer of β application servers which are then connected to γ database servers [7]. Again, the servers and the two switches have demand C and all links have bandwidth demand B.

C. Extended Problem Definition for Cloud Environments

With SDN and network programmability in mind, we do not only consider reserving virtual machines and links between them (as an overlay), but also reserving resources on programmable switches (*e.g.,* match-action rules) and virtual links forming a topology. We denote the set of physical switches as X_S and the set of virtual switches for request i as X_{R_i}. Thus, in our formulation, we do need at least three different mapping functions: $f_V : V_{R_i} \to V_S$, $f_X : X_{R_i} \to X_S$, and $f_E : E_{R_i} \to E'_S$, respectively, where $E'_S \subseteq E_S$ is a path through S consisting of multiple edges $e \in E_S$. Furthermore, we extend the scalar capacities and demands of resources and allow capacity and demand vectors that represent capacities and demands of different types, *e.g.,* CPU cycles, GPU cycles, memory, storage, etc. Thus $c_i(s)$ and $d_i(r)$ refer to capacity and demand of type i for physical resource s or virtual resource r, respectively.

D. VNE for Data Centers

As explained in Section II, in our study, most existing approaches fell short with respect to a subset or all of our requirements. Regarding the requirement to be suited to physical substrate topologies that are more commonly found in data centers, the algorithm presented by Rabbani et.al. [21] is the one which we leverage to meet this need. In their approach a minimum cost flow (MCF) formulation is used to map virtual machines onto physical servers and virtual switches onto physical switches. In order to

allow mapping multiple servers from the same request onto a single physical server, the algorithm operates in n rounds for n virtual servers in set V, where in each round only a single virtual server $v_i \in V$ is mapped to a set of possible substrate nodes $s_1, s_2, ..., s_m \in S$. In the first round S only consists of s_1. If no mapping can be found, S is extended by an adjacent server and the MCF solver runs again. The algorithm thus solves the MCF problem for server mapping $O(m \times n)$ times. The cost function is given as $a(i,j) = b(j) \times \frac{1}{|\Delta_1(j) - \overline{\Delta_1}|}$, where $b(j)$ is the used bandwidth of server j and $\overline{\Delta_1}$ is the mean of the residual capacity of type 1 (here CPU capacity). Switch mapping is carried out in a similar fashion, but bypasses the requirements of $n \times m$ rounds at the cost of only allowing one virtual switch from a request on a single physical switch. Finally, paths are mapped using an all-pair shortest path algorithm. Coordination between node and link mapping is achieved through the bandwidth parameter $b(j)$ making sure that virtual nodes are not placed on physical nodes with already exhausted upstream bandwidth. Although, this is not a full guarantee that the link mapping will succeed (as higher link layers may already be fully utilized). This heuristic works well in our experiments and achieves sufficiently high acceptance rates and network utilization.

E. Expansion and Contraction of Networks

In a virtualized infrastructure, scaling to meet higher demands typically means to horizontally scale the network, which can be as simple as adding new virtual machines that are connected to the existing part of the cluster. At the initial deployment of a new service or product, often a couple of tens of connected VMs are sufficient to serve early adopters or test users. As an application matures, services need to cope with substantial increases in requests over time. For this, cloud services often use setups that are able to automatically predict workloads and scale their infrastructure using their cloud provider's API ahead of time in order to avoid any performance drops or bottlenecks. Depending on the desired granularity, this means that virtual networks need to scale often, sometimes several times per day [17], [20].

As most existing VNE work only allows embedding or removing complete networks, we introduce two new primitives: *expand* and *contract*. We extend the above described embedding scheme with these new primitives.

Expand The *expand* operation takes as arguments the virtual network to be expanded, the network topology to be added to the network, and a set of constraints that define what link ends of the new topology need to be connected, and where, to the existing topology. We elaborate on the realization of this primitive further below.

Contract The *contract* operation removes a list of nodes from a specific network including all links that are not needed anymore. We explain this more in section III-E2.

1) Expand Networks: Conceptually, the expansion of a network is the embedding of a superset V' of the existing network with additional placement constraints. Specifically, each existing node is annotated with a constraint such that it may only be placed on the physical node to which it is currently mapped. Additionally, the constraint set includes the constraints of how the new nodes $(V' - V)$ should be connected to the existing nodes V.

In the modified bipartite minimum cost flow formulation for expansion, the only egress edge from an existing node leads to the physical node to which it is currently mapped. However, this comes with the complication that additional capacity must be added to the existing physical nodes in order to allow for the concurrent mapping of V and V', if V is not unmapped before running the algorithm for V'. Alternatively, V can be internally unmapped before solving the expansion MCF program. In either way, the expansion must be carried out as a transaction, making sure that if the expansion algorithm is unable to expand the network, the original network V is still mapped and all node and link capacities are reset.

In addition, the new nodes should be placed as close to the existing nodes as possible. As a result, this distance must be taken into account in the objective function. We adapt the cost function to take into account the proximity to the existing part of the network, such that

$$a(i,j) = b(j) \times \frac{1}{|\Delta_1(j) - \overline{\Delta_1}|} + \delta(i,j), \qquad (1)$$

where $\delta(i,j)$ is the distance (in terms of hops) from the closest (in terms of hops) node to i that is part of the original network if i was placed on j.

2) Contract Networks: Contracting a network is achieved by removing a subset V' from V and freeing the resources in the physical network. It is crucial, however, that the nodes (and links) in V' do not have the effect of partitioning V into multiple parts leaving the network in an inconsistent state. This means that the subset of nodes to be deleted must be carefully chosen and cannot just be sampled randomly, which is guaranteed in our solution.

F. Implementation and Cloud Simulator

We implemented our virtual network embedding algorithm using the C++ bindings of the Gurobi Linear Program Solver [1]. As an illustration of the runtime of this algorithm, consider a topology with four spine switches, 24 leaf switches, and 16 servers per leaf, resulting in a network with 28 switches and 384 servers. Using a network of this size with the described properties we achieve embedding times roughly between 0.1 and 15 seconds and expansion times between 3 and 40 seconds on a 3.1 Ghz Core i7 processor with 16GB of memory.

IV. Resource Fragmentation

As alluded to in Section I, embedding, removing, and scaling virtual networks causes the network to become fragmented. This is analogous to fragmentation on a hard drive. Fragmentation does not only cause rejections of virtual network requests, but also negatively affects application performance and InP efficiency through long paths between connected VMs.

A. Motivating Examples

We show the effects of fragmentation in two examples.

First consider the network from Figure 1. For this example, we ignore link and switch capacity. While the expansion request of the red network can be accommodated (see Figure 1b), this mapping results in a distance

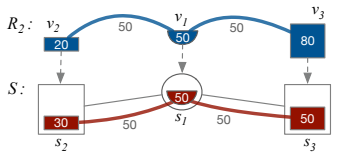
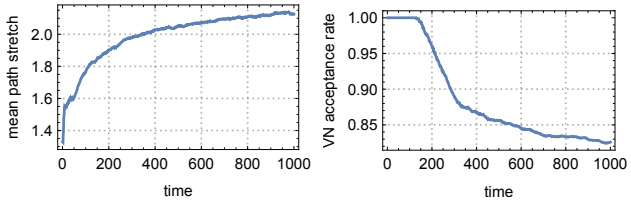


Fig. 3: Fragmented Substrate S with pending request R_2



(a) Increasing Path Length (b) Decr. VN Accept Ratio

Fig. 4: Fragmentation over Time

inflation between the switch (round icon) and the new server (square icon) of a factor of 3, negatively impacting delays along this path. After defragmentation each virtual link again corresponds to one and only one physical link.

Second, consider a substrate network S consisting of one switch s_1 connected to two servers s_2, s_3 over links e_1 and e_2. A virtual network request R_1 of the same topology has been embedded onto this network utilizing 30%, 50%, and 50% of the nodes, as well as 50% and 50% of the links. A second VN request R_2 now needs to be embedded. The two networks are depicted in Figure 3. Even though the overall resource utilization of the substrate network is less than 50% and the overall residual capacity is sufficient to accommodate the request, R_2 would be rejected, as there is no physical node with a resource residual of $\geq 80\%$ available. Mapping both virtual servers from R_1 onto s_3 would entirely free up s_2 and R_2 could be successfully mapped. Ideally, a defragmentation algorithm in this case would do exactly that.

B. Impacts of Resource Fragmentation and Metrics

Resource fragmentation affects both the Infrastructure Provider, as well as the tenant. For the InP, a high path stretch means unnecessary utilization of network resources, including switches. In this paper, we consider switch resources to be forwawrding rules or TCAM space and not backplane bandwidth. Of course, a virtual link passing through a switch (even if label switching technologies are used to reduce forwarding state), uses resources on this switch, most notably in terms of bandwidth and power consumption. These resources could be saved using a more optimal network mapping. Figure 4a shows the increase in mean path stretch over all embedded virtual networks in a physical network with a leaf-spine topology consisting of 28 switches and 384 servers when adding, deleting, expanding, or contracting virtual networks 1000 times. These results were obtained from discrete event simulations and are averaged over 10 runs with different randomized workloads. For the tenant, an increase of path stretch means amplified variability in path latency, resulting in poor application performance, prolonged job completion times, and higher cost.

Furthermore, based on how resources are blocked throughout the physical network (as illustrated in the previous section), virtual network requests need to be rejected even though the network is not entirely utilized. This impacts the virtual network acceptance rate and thus the provider's revenue. Figure 4b shows the decline of acceptance rate over time in the above described experiment. This primarily happens when all resources are equally utilized, *i.e.,* the resource utilization variance is low (cf. second example in section IV-A).

V. Defragmenting the Network

To overcome the issue of fragmentation in virtualized data centers, instead of further improving VNE algorithms to minimize fragmentation, we propose a new network management primitive: *network defragmentation*.

A. A New Network Management Primitive

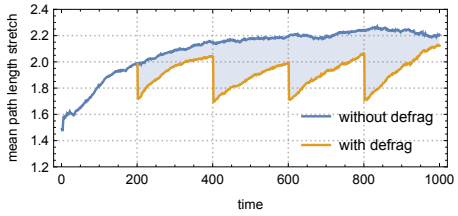
In our model, *Network Defragmentation* means to periodically run a defragmentation algorithm on the network infrastructure. This is analogous to a defragmentation program in the context of disk fragmentation. We believe this is a better approach than further improving VNE algorithms for two reasons. First, when running virtualization in a full online manner, *i.e.,* without having knowledge about the duration of virtual network requests and upcoming workloads, it is complex to customize VNE algorithms to properly account for fragmentation simply because of their inability to know about upcoming requests. This is particularly true when allowing for dynamic scaling, including expansion and contraction of networks as described earlier (section III-E). Second, with modern machine and network migration techniques and the use of centralized storage, the cost for (even large) migrations can be low; and thus presents a viable option to significantly improve network performance, efficiency and revenue.

B. Defragmentation Heuristics

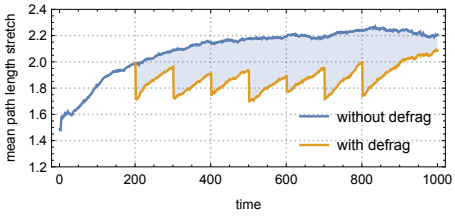
Two key questions include when to migrate, and which networks to migrate (and where to). These questions correspond to the monitor and defrag planning processes illustrated in Figure 2. In this paper we propose initial heuristics, with a goal of studying the performance benefits of defragmentation. We intend to study further optimizations as future work.

1) Monitor (when to migrate): When to perform a defragmentation is a complex question, as there is overhead in performing migration. There are generally two approaches one can follow: one that reacts to some condition in the network, and one that operates periodically. A reactive system would monitor the mappings for some condition indicating 'fragmentation' (*e.g.,* average path length or the utilization variance is low). Alternatively, a periodic system would be routinely scheduled, whether at a certain time of day, or after a certain amount of activity. For our system, we implemented such a periodic system which triggers a defragmentation after every di rounds.

2) Defrag Planning (what/where to migrate): Determining what to migrate and where to migrate it to is also a complex problem. To serve as bounds on performance, in this paper, we perform a complete remapping. That is, we unmap all virtual networks, sort by network size (biggest to smallest), and re-embed each of them in order. This

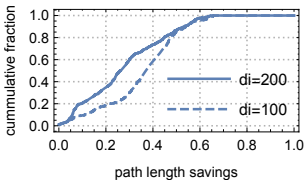


(a) Defragmentation every 200 rounds

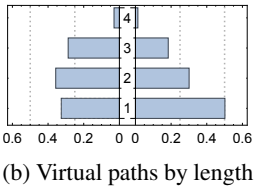


(b) Defragmentation every 100 rounds

Fig. 5: Improving application performance (reducing the mean path length) with defrag. Note the periodic sharp drops in the path length, which correspond to when a defragmentation was performed.

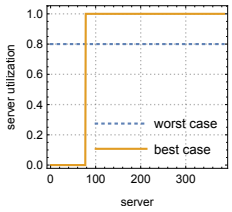


(a) Savings depending on defrag. interval

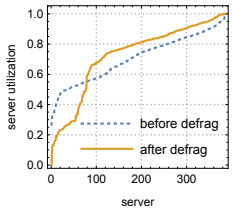


(b) Virtual paths by length before defrag. (left), and after defrag. (right)

Fig. 6: Path Length Savings



(a) optimal / worst case



(b) from simulations

Fig. 7: Server Utilization Distributions

effectively compensates for the dynamic behavior (adding, deleting, expanding, and contracting) networks, as it is reformulated as a series of add operations. Re-mapping a subset of the network (*e.g.,* the most fragmented parts of the network) is a longer term goal.

VI. Evaluation

We evaluate the efficacy of our proposal by our key fragmentation metrics: virtual path length stretch distribution and resource utilization distribution.

A. Simulation

For our experiments, we again use a leaf-spine physical topology with 384 servers and 28 switches. The requests are sampled at equal probability from the three used virtual topologies (VC, VOC, and 3T). All previously defined network parameters are sampled randomly from uniform distributions with varying parameters. For each experiment, we run 1000 rounds of events (map, expand, contract, delete) at equal probability. We run each experiment 10 times with different workloads sampled from the same distributions. Our baseline dataset contains runs of all workloads without defragmentation. In dataset 1 we remap all currently mapped networks every $di = 200$ rounds after an initialization period of 200 rounds. In the second dataset we remap every $di = 100$ rounds.

B. Virtual Path Length

We see the most promising results in reduction of virtual path lengths, the metric that mainly impacts overall network performance. Figure 5 shows the development of mean path length stretch over time for both datasets. The time steps and significant reductions at each defragmentation cycle are clearly visible. On average, when using defragmentation, we reduce the mean path length by 0.25 over baseline (defrag every $di = 200$ rounds) and 0.31 over baseline (defrag every $di = 100$ rounds), respectively. Due to the fewer physical links used, this results in link utilization savings of 5.7% and 8.1% on average. The full distributions are depicted in Figure 6a.

When looking at the individual path lengths, we can see that the number of virtual paths with the worst possible path length of 4 (diameter of our topology), is reduced by 52% on average. The number of paths with length 3 is reduced by 62% on average. Figure 6b shows the fraction of paths by length, on average, before defragmentation (left side) and after defragmentation (right side).

C. Utilization Distribution

As explained in the first example of section IV-A, in a fragmented network we can end up with a lot of partially used resources. This is opposed to a utilization distribution, where resources are either heavily utilized or lightly (or not at all) utilized. VNE algorithms tend to introduce this effect over longer runtimes (cf. Section IV-B). While it is harder to quantify this effect, the variance (denoted as σ^2) of the utilization distribution can be used as a metric. A lower variance here indicates higher fragmentation.

Graphically, we can illustrate this effect by ordering all resources by utilization (from low to high) and plotting the resulting list. Figure 7 shows two such graphs. The left graph shows the ("best") case where all resources are either equally utilized at 80% (σ^2 = 0), or either not utilized at all (20% of them) or fully utilized (80% of them). Obviously, the latter case (σ^2 = 0.16) is almost impossible to achieve, especially given that we primarily optimize for lower path length. However, the plot on the right side shows that defragmentation improves the network toward this goal, where the utilization distribution has a variance of $\sigma^2 = 0.024$ before defragmentation and doubles to $\sigma^2 = 0.048$ after defragmentation. In particular, we more than double the fraction of servers utilized by less than 50% from 9.8% before defragmentation to 20.1% after defragmentation. From an operations perspective, having more completely free resources leaves room to add new virtual networks and allows for opportunities to conserve energy by shutting off unused resources.

VII. Discussion

While we see promising results for our first steps toward a network defragmentation solution, we see two main areas for further research:

a) Fragmentation Analysis and Selective Migration: Our previously described fragmentation metrics all quantify global network properties, however during our simulations and using a network visualization tool, we saw that request rejections can be due to comparatively small and local effects, such as a single or a few highly utilized links effectively *blocking* underutilized servers or switches from the rest of the network rendering these resources useless for the VNE algorithm. For future work, we envision techniques detecting such scenarios and then performing defragmentation in just a small zone of the data center. At the cost of embedding time, such detection and mitigation could be integrated in the VNE algorithm to further improve acceptance rates.

b) Minimizing Migration Cost and Performing Migrations: Even though modern machine and network migration techniques are able to migrate resources with negligible downtime, there inevitably is a cost associated with moving virtual resources from one physical link or server to another. We showed an upper bound for what defragmentation can achieve; in future research, the cost of migration should be factored in remapping networks. This way, a trade-off can be made between migration cost and optimality of placement. Additionally, after computing a new mapping, the exact migration steps to realize this mapping must be determined. For example, migration steps can have dependencies among each other, which sometimes are circular dependencies such that resources must effectively be swapped using some scratch capacity.

VIII. Conclusions and Future Work

Elasticity is one of the key properties of cloud computing, where tenants can scale their infrastructure up-and-down based on demand. Most virtual network embedding solutions (i) do not allow any primitives for scaling an infrastructure, only creating and destroying, and (ii) provide only a static mapping, where, once mapped, the virtual to physical mappings do not change. In this paper, we introduced the concept of network expansion and contraction as a key requirement for VNE algorithms and provide a formulation for this problem. We show how operating a network in this manner amplifies the problem of resource fragmentation, which can degrade the performance of applications. In response, we introduced a new network management primitive, network defragmentation, which leverages migration capabilities to remap embeddings to optimize application performance. Through simulations, we showed that a simple network defragmentation scheme can successfully improve network performance and reduce fragmentation in the substrate network. As a next step, we will build a system to run the defragmentation. We also intend to further optimize the described defragmentation planning in two directions: determining when to perform defragmentation, and determining a minimal subset of networks that can be migrated while still achieving significant reductions in fragmentation.

IX. Acknowledgements

This work was supported in part by NSF NeTS award number 1320389, by FCT through funding of the uPVN project, ref. PTDC/CCI-INF/30340/2017, and LASIGE Research Unit, ref. UID/CEC/00408/2019.

References

[1] Gurobi Optimization. http://www.gurobi.com.
[2] Amazon. ELB. https://aws.amazon.com/elasticloadbalancing/.
[3] H. Ballani, P. Costa, T. Karagiannis, and A. Rowstron. Towards Predictable Datacenter Networks. In *Proceedings of the ACM SIGCOMM 2011 Conference*, New York, NY, USA, 2011. ACM.
[4] E. Banks. Data center network design moves from tree to leaf. http://searchdatacenter.techtarget.com/feature/Data-center-network-design-moves-from-tree-to-leaf, 2013.
[5] Cheng, X et.al. Virtual Network Embedding Through Topology-aware Node Ranking. *SIGCOMM CCR*, 41(2):38–47, apr 2011.
[6] M. Chowdhury, M. R. Rahman, and R. Boutaba. ViNEYard: Virtual Network Embedding Algorithms With Coordinated Node and Link Mapping. *Networking, IEEE/ACM Trans. on*, 20(1), feb 2012.
[7] Digital Ocean. 5 Common Server Setups For Your Web Application. https://www.digitalocean.com/community/tutorials/5-common-server-setups-for-your-web-application, 2014.
[8] D. Drutskoy, E. Keller, and J. Rexford. Scalable Network Virtualization in Software-Defined Networks. *IEEE Inet. Comp.*, 2013.
[9] C. et.al. Virtual Network Embedding for Evolving Networks. In *Proc. GLOBECOM 2010*, pages 1–5, dec 2010.
[10] V. S. et.al. Application-aware virtual machine migration in data centers. In *Proc. INFOCOM 2011*, 2011.
[11] Fajjari, I et.al. VNR algorithm: A greedy approach for virtual networks reconfigurations. In *Proc. GLOBECOM 2011*, dec 2011.
[12] A. Fischer, J. F. Botero, M. Till Beck, H. de Meer, and X. Hesselbach. Virtual Network Embedding: A Survey. *Communications Surveys Tutorials, IEEE*, 15(4):1888–1906, 2013.
[13] C. Fuerst, S. Schmid, L. Suresh, and P. Costa. Kraken: Online and elastic resource reservations for multi-tenant datacenters. In *IEEE INFOCOM 2016*, pages 1–9, apr 2016.
[14] Gao, X et.al. A new algorithm with coordinated node and link mapping for virtual network embedding based on LP relaxation. In *Asia Comm. and Phot. Conf.*, 2010.
[15] S. Ghorbani, C. Schlesinger, M. Monaco, E. Keller, M. Caesar, J. Rexford, and D. Walker. Transparent, Live Migration of a Software-Defined Network. In *Proc. SOCC 2014*, SOCC '14, 2014.
[16] Guo, C. et.al. SecondNet: A Data Center Network Virtualization Architecture with Bandwidth Guarantees. In *Proc. CoNext '10*.
[17] Hudl. Queuing Up Heavy Tasks to an Autoscaling Worker Farm. http://public.hudl.com/bits/archives/2014/11/12/queuing-up-heavy-tasks-to-an-autoscaling-worker-farm/, 2014.
[18] Koponen, T. et.al. Network Virtualization in Multi-tenant Datacenters. In *Proc. NSDI 2014*, Seattle, WA, apr 2014.
[19] L. Chen et.al. Cache contention aware virtual machine placement and migration in cloud datacenters. In *Proc. ICNP 2016*, 2016.
[20] Netflix. Scryer. https://tinyurl.com/y7ypvxkd.
[21] M. G. e. a. Rabbani. On tackling virtual data center embedding problem. In *2013 IFIP/IEEE International Symposium on Integrated Network Management (IM 2013)*, pages 177–184, may 2013.
[22] Rabbani, M.G. et.al. On tackling virtual data center embedding problem. In *2013 IFIP/IEEE IM*, may 2013.
[23] Rost, M. et.al. Beyond the Stars: Revisiting Virtual Cluster Embeddings. *SIGCOMM Comput. Commun. Rev.*, jul 2015.
[24] Sherwood, R. et.al. Can the Production Network Be the Testbed? In *Proc. OSDI 2010*, Berkeley, CA, USA, 2010. USENIX.
[25] C. Wilson, H. Ballani, T. Karagiannis, and A. Rowtron. Better Never Than Late: Meeting Deadlines in Datacenter Networks. In *Proc. SIGCOMM 2011*, New York, NY, USA, 2011.
[26] W.-L. Yeow, C. Westphal, and U. Kozat. Designing and Embedding Reliable Virtual Infrastructures. In *Proc. VISA 2010*, 2010.
[27] M. Yu, Y. Yi, J. Rexford, and M. Chiang. Rethinking virtual network embedding: Substrate support for path splitting and migration. *SIGCOMM CCR*, 38(2):17–29, Mar. 2008.
[28] M. F. Zhani, Q. Zhang, G. Simona, and R. Boutaba. VDC Planner: Dynamic migration-aware Virtual Data Center embedding for clouds. In *2013 IFIP/IEEE International Symposium on Integrated Network Management (IM 2013)*, pages 18–25, may 2013.
[29] Y. Zhou, Y. Li, D. Jin, L. Su, and L. Zeng. A virtual network embedding scheme with two-stage node mapping based on physical resource migration. In *2010 IEEE International Conference on Communication Systems*, pages 761–766, nov 2010.