Algorithms for More Efficient Network Virtualization and Resource Utilization

Christopher Freas
Department of Computer Science
Georgia State University
Atlanta, Georgia 30303
Email: cfreas@cs.gsu.edu

Abstract—Virtualization has proven to be a must-have technology for server platforms, enabling rapid innovation, on demand computing, and flexible resource allocation. The networking world is embracing these ideas as it adopts virtualization technologies and software-defined networking (SDN). This paper examines two different, but complementary approaches to better make use of the limited resources inherent in infrastructure networks. In the first paper, an opportunistic embedding scheme for virtual networks is proposed. In the second paper, a new approximation algorithm to maximize the effective transmission rate of virtual machine (VM) migrations is proposed. While the focus of the first paper is on optimal placement of virtual network components, the second paper complements this idea by offering a novel means of effectively moving computing resources around the network.

I. INTRODUCTION

One of the key challenges infrastructure providers (InPs) face is how to best allocate their resources among their customers, who operate as service providers (SPs) to end-users. For example, they typically provide bandwidth, CPU, or storage resources at some fixed or time-based fee. A key challenge for network virtualization is the so-called virtual network embedding problem (VNE), which deals with efficiently embedding virtual networks, each with varying requirements, onto a shared substrate or physical network. Zhang et al.’s Virtual Network Embedding with Opportunistic Resource Sharing (ORS) [21] accounts for the time-varying resource requirements of VNEs instead of simply reserving fixed resources throughout the virtual network’s lifetime as all the previous approaches do.

Complementing the issues associated with network virtualization is the problem of efficiently migrating virtual machines (VMs) around an InP’s network to optimize resource usage or to provide fault tolerance and load balancing. Wang et al. argue in [16] that while much attention has been focused on optimal placement of VMs, the process of migrating VMs has been overlooked. They propose a fully polynomial time approximation (FPTA) algorithm that computes the optimal migration sequence and bandwidth utilization for each migration. Unlike the existing approaches, the goal is to maximize the overall transmission rate of VM migrations so the downtime of the VMs is dramatically lowered.

The rest of this paper will cover ORS and FPTA in depth and how they compare with each other. Section II covers ORS beginning with an overview of the virtual network embedding problem. Each algorithm proposed in the ORS paper is explored and proven in its own subsection, concluding with an examination of how these algorithms were evaluated. Section III covers FPTA in a similar manner. Section IV compares the two papers with each other. Finally, section V provides ideas for future research in network virtualization.

II. OPPORTUNISTIC RESOURCE SHARING (ORS)

A. The Virtual Network Embedding Problem

When discussing ossification of the Internet’s fundamental architecture, two competing viewpoints emerge. The first is the so-called “purist”, who views Internet evolution as an iterative process with any proposed architectural changes confined

1Service providers offer services such as voice over IP (VoIP), security, and virtual private networks (VPNs) to end-users, to name a few.

2The industry uses the terms “underlay” for the physical network and “overlay” for the virtual network.
to being singular in nature and subject to joint agreement by the Internet Service Providers (ISPs) [2]. The second view is that of the “pluralist”, who advocates a diverse platform capable of providing many services. In particular, the pluralist advocates that virtualization be an attribute of the Internet instead of a means to an end [15]. Virtualization is of particular interest to researchers, since it provides a means to experiment with new architectures through live experimentation as opposed to simulation [2].

A promising application of virtualization is in embedding virtual networks on top of physical networks. This problem is known as the virtual network embedding problem and has been shown to be NP-Complete by a reduction from the multiway separator problem to this problem [1]. A more formal definition of this problem is given in the next section.

B. The Traditional Network Embedding Process

The process of embedding a virtual network onto a substrate network (described in [21]) is modeled as a weighted, undirected graph $G^s = (N^s, E^s)$ representing the substrate network (with $N^s$ and $E^s$ representing the sets of substrate nodes and links, respectively) and $G^v_i = (N^v_i, E^v_i)$ representing the proposed virtual network. Because the substrate network will support more than one embedding, virtual networks and their components are indexed by $i$. Each substrate node $n^s\in N^s$ is associated with a CPU capacity $C(n^s)$, and each substrate link $e^s = (n^s_i, n^s_j) \in E^s$ is associated with a bandwidth capacity $B(e^s)$. The residual resources of each $n^s$ and $e^s$ are denoted by $RC(n^s)$ and $RB(e^s)$, respectively. The residual resources of each are computed differently depending on whether ORS is used or not, since ORS takes a more granular view of resource allocation than all preceding works in the literature.

Figure 1 shows a substrate network with each node and link’s resources and Figure 2 shows a virtual network with each virtual node and link’s resource requirements.

The set of loop free paths from $n^s_i$ to $n^s_j$ is denoted as $P^s(n^s_i, n^s_j)$. The virtual network in Figure 2 could be embedded onto the substrate network in Figure 1 with the following set of node mappings

\[
\{a \rightarrow B, b \rightarrow F, c \rightarrow G, d \rightarrow H\}
\]

and the following set of link mappings \{(ab) → BD, DF, (ac) → BG, (bd) → FH, (cd) → GH\}.

Given this mapping, the virtual network embedding problem can be formulated as a decision problem and is defined in [10] as follows:

**Given the virtual network** $G^v = (N^v, E^v)$, **and substrate network** $G^s = (N^s, E^s)$, **can the virtual network be mapped to the substrate network while satisfying the following requirements:** (i) node mapping requirement: one virtual node is mapped onto one substrate node, with no two virtual nodes (from the same virtual network) sharing the same substrate node; (ii) link mapping requirement: one virtual link is mapped to path(s) between the two substrate nodes which hold the two virtual nodes of this virtual link; (iii) capacity constraint: for each virtual node/link, it is mapped to the substrate node/path(s) with sufficient computing/bandwidth capacity?

The InP must decide if the proposed mapping should be accepted or rejected, and will have to do the same for future embedding requests based

---

3The multiway separator problem deals with partitioning a graph into $k$ groups of similar size by removing a minimal number of edges.

---

This work is licensed under a “CC BY-SA 4.0” license.
upon remaining resources. In the case of this proposed embedding, the virtual network’s allocation of substrate resources exists throughout its lifetime. Because this is the case for all previous approaches, a summary of the existing approaches follows.

C. Existing Approaches to the Virtual Network Embedding Problem

Existing approaches to solving the virtual network embedding problem are varied in their details but all involve using heuristics to approximate a solution. In the interests of terseness, only the relevant aspects of a few of the previous approaches are discussed and it can be safely assumed that there is a fair amount of overlap between them.

Ricci et al. propose in [14] the use of simulated annealing as a heuristic. Their solver, called assign, implements simulated annealing. They complement assign with three additional functions - init_score to score any violations in resource constraints, add_node, which takes a substrate network and a virtual network and computes changes in resources and violations, and remove_node, which simply performs the inverse of add_node. In order to perform path computations, assign makes use of Dijkstra’s shortest path algorithm and penalties are assigned based on the number of nodes and links an embedding uses. In order to reduce the size of the search space, assign uses equivalence classes to partition nodes when they have identical types and features. The resulting equivalence classes are each called a pclass and each pclass is embedded instead of mapping nodes directly.

Lu et al. propose in [13] the use of constraint-based network design using pairwise constraints as well as termination constraints and distance constraints. The topology of the network is then constrained to a backbone-star topology, where some nodes are designated as backbone nodes, and the remainder are star nodes that connect only to backbone nodes. Linear programming with relaxation is used to arrive at an embedding. The authors conclude that as the network becomes more “tree-like”, the cost of the backbone topology is reduced.

Zhu et al. argue in [23] that requests for virtual network embeddings need to be handled efficiently and on demand. Efficiency is defined as achieving a low and balanced load on the substrate network and on demand handling is defined as handling each VN request based on the current state of the network. Like most of the literature, assignment is based on linear programming, however, a variable called stress ratio is used to improve load balancing. The authors propose two versions of their algorithm - one that does assignment without reconfiguration (i.e. an offline algorithm) and one that can dynamically handle incoming VN requests and perform embeddings.

Chowdhury et al. propose in [6] two online embedding algorithms - D-ViNE (Deterministic Virtual Network Embedding) and R-ViNE (Randomized Virtual Network Embedding), which use deterministic and randomized rounding techniques, respectively, to approximate the values of the original mixed integer program. Integer constraints are relaxed in the linear program in order to allow it to be solved in polynomial time. Flows between nodes are accounted for using binary variables. A third algorithm, WiNE (Window-based Virtual Network Embedding) adds lookahead capabilities to the previous algorithms by doing batch processing of requests. By doing this, embedding requests can then be accepted based on their revenue potential to the InP.

D. ORS - Motivation and Overview

ORS was motivated by the key observation that none of the previous approaches account for the time-varying resource requirements of VN embeddings. Since virtual nodes and links do not consume all of their allocated resources all the time, the unused resources could instead serve other VN requests. Thus, in addition to finding near optimal embeddings of nodes and links using approximation techniques as the previous approaches do, ORS accounts for time-varying resource requirements using a probabilistic model. The ORS model assumes that the substrate network employs time-division...
where time is composed of frames and each frame further consists of time slots. The number of time slots in each frame depends on the physical bandwidth of the substrate link. Thus, several virtual links can be embedded into a substrate link, the volume of which depends on the number of time slots available in each frame.

### E. Embedding Process

The model for embedding with ORS consists of two components: a macro-level component for basic node-to-node and link-to-link embedding and a micro-level time slot assignment component. The macro-level component uses a strategy similar to [20] where nodes are placed in a queue. ORS differs in that it places nodes in its queue by decreasing number of required time slots instead of decreasing revenues.

In order to capture the varying requirements of the embedding, the traditional $C(n^s)$ and $B(e^s)$ node and link requirement representations are replaced with tuples $(b(n^v), v(n^v), p(n^v))$ and $(b(e^v), v(e^v), p(e^v))$, respectively, where $b()$ represents a basic subrequirement, $v()$ represents a variable subrequirement, and $p()$ represents the probability that the variable subrequirement occurs. A basic subrequirement exists throughout the lifetime of the embedding and a variable subrequirement is ephemeral. The queue representation is then $(b(Q[i]) + p(Q[i])) + v(Q[i])$ for virtual node $Q[i]$. Each virtual node is then simply mapped from the head to the tail of $Q$ to the unused substrate node with the most available resource. If a substrate node is not able to be found for $Q[i]$ to be mapped to, the VN embedding request is rejected. Each virtual link is embedded in a similar fashion to the shortest path with enough bandwidth between the substrate nodes to minimize the span of the embedding. In the case of virtual nodes with multiple paths, the $k$ shortest paths are computed to minimize the span between the underlying substrate nodes of the embedding.

Time-division multiplexing encodes multiple signals in a transmission line such that each signal appears a fraction of the time in the line.

Revenue is defined as the economic potential of accepting VN embedding requests [20].

The $k$ shortest-paths problem is similar to the single source shortest paths problem; that is, you could modify Dijkstra’s algorithm or the Bellman-Ford algorithm to compute more than one path between source and destination nodes.

1) **Macro-level Assignment:** The macro-level assignment algorithm previously described is detailed formally in Algorithm 1 in [21] and, using the graph notation described previously, has a time complexity of $O([N^v]log([N^v]) + |N^v|)$. If multiple paths exist in a VN request, finding the $k$ shortest paths has complexity $O([N^s]^2 + [N^v]log([N^s]) + k)$. Thus, the total time complexity is $O(([N^v]log([N^v]) + |N^v|) + [N^s]^2([N^s] + [N^v]log([N^v]) + k)) = O([N^s]^4)$.

2) **Micro-level Time Slot Assignment:** The micro-level time slot assignment takes the substrate nodes and links for the embedding of $G^v$ and runs either First Fit by Collision Avoidance (CFF) or First Fit by Expectation of Indicators’ Sum (EFF) to determine time slot assignments and update the residual resources of substrate nodes and links.

The authors represent the maximum variable subrequirement by $max(v)$ and the maximum capacity of the substrate links and nodes by $max(max(B), max(C))$. Then $F = max(v) \times max(max(B), max(C))$ and the CFF and EFF components of the micro-level time slot component have time complexity $O(F)$. The micro-level component then has complexity $O(F|N^s|^2)$ and ORS has overall complexity $O([N^s]^4 + F|N^s|^2)$.

Due to the opportunistic nature of the micro-level time slot assignment, collisions become a problem. Since the basic requirements of the VN are assigned in a greedy fashion, the authors limit the scope of the discussion of collisions to the variable subrequirements. Collisions between variable subrequirements are then defined as multiple units of subrequirements occurring simultaneously. Collisions (and utilization) are constricted by the use of a collision threshold, $p_{th}$, which represents a synthetic “volume” of a time slot.

The model for collisions is probabilistic, which the authors represent by the following:

$$Pr(D_j) = Pr\left[\sum_{i \in D_j} X_i \geq 1\right] =$$

$$1 - \prod_{i \in D_j} (1 - p_i) - \sum_{i \in D_j}(p_i \prod_{k \in D_j,k \neq i} (1 - p_k))$$

In the model, $D_j$ represents the set of variable subrequirements that substrate time slot $j$ ($t(s_j)$) is assigned to. $X_i$ indicates whether the $i$th variable subrequirement occurs (that is, $Pr[X_i = 1] = p_i$).

Details of CFF and EFF are in sections F.2 and F.3, respectively.
Hence, the probability of a collision between the elements in the set $D_j$ of variable subrequirements in substrate time slot $ts_j$ is the probability that the sum of the indicators of each variable subrequirement $i$ is greater than or equal to 1. Equivalently, the probability of a collision between the elements in the set $D_j$ of variable subrequirements in substrate time slot $ts_j$ is the probability of no collisions between like elements of the same set of variable subrequirements $i$ as well as the probability of no collisions between distinct elements of variable subrequirements (i.e. the variance of the elements). This motivates a formal definition of the time slot assignment problem, described next.

F. The Time Slot Assignment Problem (TSA)

The authors define the TSA problem as follows:

Given a set of $n$ virtual links from different VNs, the variable subrequirement of the $i$th virtual link is $v_i$ time slots, each of which is needed with probability $p_i$. Find an assignment of substrate time slots to the subrequirements to minimize the number of slots used, such that: 1) for the variable subrequirement of the $i$th virtual link, the number of time slots assigned to it is at least $v_i$; and 2) the collision probability at each substrate time slot is no more than a given collision threshold $p_{th}$.

![Example TS Assignment](image)

Fig. 3. A feasible time slot assignment.

An example of this is shown in Figure 3. In the figure, the time slots $ts_1 \ldots ts_n$ are at the bottom and the probability threshold $p_{th}$ is 0.2. The number of time slots shown in the figure is arbitrary; in practice, a greater number of time slots would exist and would constitute a substrate link frame as described in the ORS overview section. Above the time slots are three sets of variable subrequirements with their associated probabilities.

Some important properties of the micro-level time slot assignment algorithm are illustrated in the figure. First, notice that no variable subrequirements have overlapping assignments (e.g. for the first variable subrequirement, requirements 1 and 2 use different time slots), a function of the time-division multiplexing nature of ORS as well as the formal decision problem. Next, observe that variable subrequirements one and three cannot share any time slots since their collision probabilities exceed the collision threshold $(0.6 \times 0.4 = 0.24 > p_{th} = 0.2)$. Subrequirements one and two can share time slots since their collision probabilities $(0.4 \times 0.3 = 0.12)$ do not exceed the threshold. Subrequirements two and three can share time slots for the same reason.

Due to the combinatorial nature of the TSA problem, the authors provide the following theorem:

**Theorem 1.** TSA is NP-hard in the strong sense.

The proof of this theorem begins with a brief introduction to the 3-partition problem and proceeds with a reduction from the 3-partition problem to the TSA problem. Feasible assignments of variable subrequirements are created such that there are $3m$ subrequirements that occur with probability $p_i$. The proof proceeds by establishing an upper bound for the collision probability, which is proved to be within $(0, 1)$. A lower bound is established by proving that any set of four subrequirements occurs with a probability strictly above the probability threshold, $p_{th}$. The remainder of the proof establishes that a valid partition of subrequirements into $m$ subsets must have exactly three subrequirements. The strongness of TSA is proved by bounding all subrequirements by a polynomial (the collision probability).

1) An Integer Linear Programming-based Optimal Solution: Inspired by the cutting stock prob-

12The proofs of all the theorems presented in the paper are offered as a supplemental paper. I can provide this upon request.

13The 3-partition problem is to decide whether a set of positive integers $S$ can be partitioned into subsets such that each subset has the same sum. The set $S$ contains $n = 3m$ positive integers, so each subset then contains $m = \frac{n}{3}$ integers. If $B$ is the sum of every subset $S_i$, the problem is NP-Complete when $\forall s \in S, \left( \frac{1}{2} < s < \frac{B}{2} \right)$
lem14 the authors explore the creation of a solution to the TSA problem by creating a integer linear programming-based solution. A pattern is denoted as a set of variable subrequirements whose collision probability is no more than the collision probability threshold, \( p_{th} \). The objective then is to minimize the number of times any given pattern can appear in a feasible assignment. The problem with this formulation is twofold: first, the number of possible patterns can be large, and second, the optimal solutions must be found by an exhaustive search. Due to these shortcomings, the authors present two possible solutions, described next.

2) First Fit by Collision Avoidance (CFF): CFF was inspired by the authors’ realization that the TSA problem is a special case of the bin packing problem15. Due to this, the first-fit algorithm16 for bin packing is employed and modified to measure the probability of collision between the set of sub-requirements \( D_j \) assigned to time slot \( ts_j \). The algorithm is formally detailed in Algorithm 2 in [21].

An approximation ratio bound is found by proving the following theorem:

**Theorem 2.** \( S_{eff} \leq S_{opt}(v_{max} \times vol_1)/(v_{min} \times vol_2) \), where \( vol_1 \) and \( vol_2 \) are the roots of equations:

\[
1 - (1-p_{min})^{vol_1} - v_{min} \times p_{min} \times (1-p_{min})^{vol_1-1} = p_{th}
\]

\[
1 - (1-p_{max})^{vol_2} - v_{max} \times p_{max} \times (1-p_{max})^{vol_2-1} = p_{th}
\]

In this theorem, \( S_{eff} \) denotes the assignment results of CFF and \( S_{opt} \) denotes the optimal solution. The authors reuse \( S_{eff} \) and \( S_{opt} \) to denote the number of assigned and optimal substrate slots used, respectively. Let \( p_{min} = min(1 \leq i \leq n)p_i \), \( v_{min} = min(1 \leq i \leq n)v_i \) and \( p_{max} = max(1 \leq i \leq n)p_i \), \( v_{max} = max(1 \leq i \leq n)v_i \).

The proof of this theorem is by cases and considers a particular but arbitrarily chosen sub-flow, \( S_I \), which requires \( d_{min} \) time slots and occurs with probability \( p_{min} \). The proof proceeds to establish a lower bound by determining the maximal allowable number of sub-flows that a substrate slot can be assigned to in this case, which is denoted by \( vol_f \). For \( n \) sub-flows, the number of substrate slots required is \( S_I = (n \times d_{min})/vol_f \). The upper bound is determined in a similar fashion using a particular but arbitrarily chosen sub-flow, \( S_{II} \). The maximal allowable number of sub-flows that a substrate slot can be assigned to in this case is denoted by \( vol_{II} \). For \( n \) sub-flows, the number of substrate slots required is \( S_{II} = (n \times d_{max})/vol_{II} \). The following inequality illustrates the computed bounds:

\[
0 < S_I \leq S_{opt} \leq S_{eff} \leq S_{II}
\]

The approximation ratio bound then follows:

\[
\frac{S_{eff}}{S_{opt}} \leq \frac{S_I}{S_f} = \frac{d_{max} \times vol_f}{d_{min} \times vol_{II}}
\]

3) First Fit by Expectation of Indicators’ Sum (EFF): By replacing the function to compute the collision probability between the set of variable sub-requirements \( D_j \) assigned to time slot \( ts_j \) with an expected value sum, the complexity of the TSA problem could be greatly reduced [21]. Let \( X_i \) be the indicator for the \( i \)th variable subrequirement. The problem then is to find, for a given \( p_{th} \), a corresponding value, \( \mu_{th} \), which the sum of the indicators of the set of variable subrequirements does not exceed. The authors call this new algorithm “first fit by expectation of indicators’ sum”.

The value of \( \mu_{th} \) is based on the following theorem:

**Theorem 3.** If \( E \left[ \sum_{i \in D_j} X_i \right] \leq \mu_{th} \), then \( Pr \left[ D_j \right] \leq p_{th} \), where \( \mu_{th} e^{1-\mu_{th}} = p_{th} \), and \( e \) is the exponential constant.

This theorem simply guarantees that if the expected value of the sum of the indicators for a set of variable subrequirements \( D_j \) is no more than \( \mu_{th} \), the collision probability of that same set of variable subrequirements \( D_j \) will not exceed the collision threshold, \( p_{th} \). The proof of this theorem uses a Chernoff bound17 to establish this, where the moment is \( \mu = E[Y] \), \( Y = \sum_{i \in D_j} X_i \). The value

14The cutting stock problem is the problem of cutting standard sized pieces of some sort of stock material into varying sizes while minimizing the wasted material. The knapsack problem can be reduced to the cutting stock problem, thus the cutting stock problem is NP-Complete.

15The bin packing problem seeks to minimize the number of bins used to store material of various sizes.

16First fit considers items to bin pack in an arbitrary order (i.e. it simply takes the current item and places it in the first available bin that can accommodate it without regard to the remaining items) and is a greedy online approximation algorithm.

This work is licensed under a “CC BY-SA 4.0” license.
of $\mu_{\text{th}}$ is established using polynomial interpolation. The theorem, along with the TSA problem, gives rise to the Expectation-based Time Slot Assignment Problem (ETSA), defined as:

Given a set of $n$ virtual links from different VNs, the variable subrequirement of the $i$th virtual link is $v_i$ time slots, each of which is needed with probability $p_i$. Find an assignment of substrate time slots to the subrequirements to minimize the number of slots used, such that: 1) for the variable subrequirement of the $i$th virtual link, the number of time slots assigned to it is at least $v_i$; and 2) the expectation of the sum of the indicators of a set of variable subrequirements that a substrate slot is assigned to is no more than a given expectation threshold $\mu_{\text{th}}$.

The complexity class of the ETSA problem is described in the following theorem:

**Theorem 4.** The ETSA problem is NP-complete.

The proof of this theorem is by a reduction from the bin-packing problem to the ETSA problem. The bound $\mu_{\text{th}}$ is the bin size and $p_1, p_2, \ldots, p_n$ are the sizes of $n$ items. An important observation is that by using EFF, the number of substrate slots that can be assigned to a set of variable subrequirements decreases and is described by a relaxation gap, $\lambda$. This relaxation gap replaces the $\mu_{\text{th}}$ notation with $\lambda \mu_{\text{th}}$ and is described further in the results section.

**G. Rearrangement**

In the course of opportunistic placement of virtual network requests, the substrate resources will tend to be underutilized. The authors propose a rearrangement protocol in order to deal with this. The protocol works by running CFF or EFF on all existing embeddings on decreasing $j$ from $N$ to 1, where $j \in D_j$ and $N$ is the total number of substrate slots. This loop ends when a substrate slot is encountered that has had a new variable subrequirement assigned to it by this protocol. The protocol can be run upon a VN request’s leave or at periodic intervals set by the InP.

**H. Results and Conclusions**

In order to quantify the performance of ORS in general and CFF and EFF in particular, the performance of a single substrate link is checked first and then the entire substrate network’s performance is measured.

1) **Single Substrate Link:** In the case of a single substrate link, the probability collision threshold $p_{\text{th}}$ is set to 0.1 and the maximum variable subrequirement $v_{\text{max}}$ is set to 10. The impact of an increasing number of variable subrequirements $n$ is linear with the number of substrate slots consumed. Even with running EFF with varying relaxation gaps, CFF still consumed fewer substrate slots. Similar results were obtained when varying the value of $v_{\text{max}}$. The running times of CFF and EFF showed that EFF had better performance as the number of variable subrequirements increased. Increasing the relaxation gap widened this performance gap substantially.

2) **Entire Substrate Network:** In the case of the entire substrate network, ORS was compared to [6] and [20]. Substrate capacities for CPU and bandwidth were generated uniformly between 50 and 100. A uniform distribution of virtual networks between 2 and 10 with each virtual node connecting with a probability of 0.5 was created. The probability collision threshold $p_{\text{th}}$ is set to 0.1 throughout the simulation and the results are averaged over 100 runs. The metrics used for performance comparisons are acceptance ratio, the number of accepted VNE requests to all requests, node utilization ratio, the amount of allocated CPU requests to overall CPU resources in the substrate network, and link utilization ratio, the ratio of the amount of the allocated bandwidth resources to overall bandwidth resources in the substrate network.

ORS is shown to have a larger acceptance ratio compared to the other embedding algorithms, as well as higher link and node utilization ratios. It is interesting to note that as time goes on, the differences between acceptance ratios become increasingly insignificant. A similar result is observed for node and link utilization ratios as density increases. This suggests that ORS undoubtedly achieves a better VNE request acceptance ratio while the improvement in utilization of nodes and links might be less pronounced.
III. VIRTUAL MACHINE MIGRATION

The problem of migrating running virtual machines (VMs) has seen much attention given to optimal VM placement but relatively little attention given to the actual migration process [16].

A. Existing approaches to VM migration

Existing approaches to VM migration vary in their details and optimization objectives but all share one common shortcoming: no migrations are done in parallel, which significantly increases overall migration times. In the interests of terseness, only the relevant aspects of a few of the previous approaches are summarized.

Zhani et al. propose in [22] VDC (Virtual Data Center), an abstraction of a data center's components (e.g. servers, routers, and switches) into an embedding scheme similar to those described in the summary of the ORS paper. Similar to the TSA problem described previously, the VDC embedding problem is proven to be NP-Hard by a reduction from the bin-packing problem to the VDC embedding problem. A migration-aware embedding framework, VDC Planner, is introduced which handles initial VDC embedding and dynamic consolidation of embeddings and is analogous to the rearrangement protocol described in section II.G. VDC attempts to use migration as a way to enforce service-level agreements (SLAs) and quality of service (QoS) requirements.

Wood et al. propose in [19] Sandpiper, a proactive black-box and gray-box framework that employs hotspot detection and mitigation to drive VM migration decisions. Hotspot data is fed into a Migration Manager component, which decides when to perform migrations, where to migrate the overloaded VMs, and importantly, how many additional resources to allocate to the migrated VMs. Migrations are then performed using a greedy heuristic, mapping the most overloaded VMs before mapping the least.

Li et al. consider load balancing a cluster of VMs in [12] and propose an affinity-aware and workload-aware migration strategy. No new algorithms are proposed; instead, the purpose of the paper is to study the effects of migrating VMs with different affinity and workload types in clusters of VMs. Optimization rules for migrations in load balanced scenarios are then established. The authors suggest a forthcoming paper where they will develop more rigorous methods and mathematical models.

B. System Overview

An illustration of the high-level system described in [16] is shown in Figure 4, which appears in [16] as Figure 1.

As can be seen in the figure, the network side is under the control of the SDN controller, allowing for programmatic control of network devices (e.g. switches and routers) using the OpenFlow protocol. The entire virtual machine environment, including the virtual machine hosts, is under the control of the SDN controller.

---

18 Running or “active” virtual machines are often called “live” virtual machines; thus the process of moving them between physical servers is often referred to as “live migration”.

19 A virtual machine (in the context of this paper) is a software abstraction of a physical computer; that is, it emulates the hardware, including BIOS, CPU, memory, and peripherals of an actual computer.

20 In the context of their paper, black-box means operating system agnostic and gray-box means that Sandpiper’s monitoring component can gather some operating system statistics from running VMs to make migration decisions.

21 Hotspot detection is defined as using monitoring and profiling of CPU, memory, and network resource utilization to ensure they do not exceed a performance threshold defined in a service-level agreement.

22 In this context, affinity means that a workload is placed on a pair of VMs and they are bounded to using the same host CPU.

23 SDN, or Software Defined Networking, is an abstraction of the control plane of networking devices into a centralized controller with a global view of the network. This allows for programmatic control of the entire network, including device configurations.

24 OpenFlow, defined by the Open Networking Foundation, allows for programmatic control of network traffic based on several network flow fields.
the OpenStack cloud controller. The Coordinator is the result of the authors’ work and gathers data from the SDN and OpenStack controllers in order to output a VM migration plan. This plan consists of a sequence of VMs to be moved along with the bandwidth that each should use for migration. This is realized by reconfiguring the network to provide a bandwidth guarantee and instructing OpenStack  to perform the migrations according to the plan.

1) Virtual Machine Migration Process: The process of migrating a live VM consists of moving the contents of the VM’s memory, disk, and network state from a source VM hypervisor to a target VM hypervisor. Copying the contents of the VM’s memory can be performed using either pre-copy memory migration or post-copy memory migration. Since and all of the existing work utilize pre-copy migration, post-copy migration is not discussed.

In pre-copy memory migration, the work of migrating the contents of memory is done in two distinct phases, the warm-up phase and the stop-and-copy phase. During the warm-up phase, pages of memory are copied from the source hypervisor to the target hypervisor while the VM is still running. Dirty memory pages are re-copied until they exceed the page dirty rate . The page dirty rate is measured by the hypervisor, which compares the pages of memory in shadow page tables it maintains to page writes by the VM to obtain the page dirty rate . During the stop-and-copy phase, the VM is stopped on the source hypervisor and any remaining dirty pages are copied to the target hypervisor. It is at this stage that VM downtime (i.e. lack of reachability) occurs.

Traffic measurement is performed by the SDN controller with the aid of an extra field in the controller’s forwarding table that carries the traffic level at each “hop” in the network. The SDN controller sends this information to the Coordinator to guide VM migration sequence decisions.

C. Motivation for Multiple Migrations

The need for multiple migrations was motivated by the fact that all previous approaches assume one-by-one migration of VMs instead of concurrent migrations . The authors also consider network utilization during migration. The result is that migrations are done in parallel and multiple network paths are allowed to be used for VM migration. It is assumed that a separate migration network does not exist, thus network utilization during migration is maximized in order to reduce the overall migration time for the sequence of VMs. The mathematical model for this is described in the next section.

D. Mathematical Model for Live Migrations

Let represent the memory size of the virtual machine, the page dirty rate during the migration, and the bandwidth allocated for the migration. Assume that memory is copied in rounds as described previously. The number of pages copied during each round is denoted by . All memory pages are copied during the first round, thus we have . At each subsequent round, dirty pages (i.e. pages that were modified during the previous round) are copied. The transmitted data at each round can be calculated as . The time spent during each round can be calculated as . Intuitively, this means that each round after the first round will be used to re-copy as many dirty memory pages as can be transmitted by the allocated bandwidth.

Let be the ratio of the page dirty rate and allocated bandwidth for the migration. The total migration time is then:

\[ T_{mig} = \sum_{i=0}^{n} T_i = \frac{M}{L} \times \frac{1 - \lambda^{n+1}}{1 - \lambda} \]

In this definition, the total time to copy memory pages consists of the ratio of initial memory copied divided by the allocated bandwidth, all multiplied by the geometric sum of page re-copies. The total number of rounds is represented by the following:

\[ n = \lceil \log_{\lambda} \frac{V_{thd}}{M} \rceil \]
Let $V_{thd}$ represent the threshold value of any remaining dirty memory pages that should be copied during the last iteration. Thus we have the inequality $V_n \leq V_{thd}$. The downtime incurred at the stop-and-copy phase is represented by $T_{down} = T_d + T_r$, where $T_d$ represents the time spent copying any remaining dirty memory pages and $T_r$ represents the time spent restarting the VM on the target hypervisor.

### E. Problem Formulation

The problem of VM migration can be represented formally as follows. Let the graph $G = (V, E)$ represent the network where $V$ represents the set of network nodes and $E$ represents the set of network links. Let $c(e)$ represent the residual bandwidth capacity of the link $e \in E$. Migrations are then represented in a tuple $\langle s_k, d_k, m_k, r_k \rangle$ where $s_k$ is the source node, $d_k$ is the destination node, $m_k$ is the memory size, and $r_k$ is the page dirty rate. Then there are $K$ migrations in total. For each migration $k \in K$, let $l_k$ represent the bandwidth allocated for the migration, $P_k$ the set of paths between $s_k$ and $d_k$, and $x(p), p \in P_k$ the flow in path $p$. Finally, since migrations contained in a migration plan will start at different times, let the variable $X_k$ indicate whether migration $k$ has started.

The authors make an important distinction between their work and that of previous approaches. Because they allow multiple migrations to occur simultaneously, the optimization objective can not be to simply minimize the down time of the migrations (i.e. the stop-and-copy phase). Thus, the volume of data transferred during each round can be simplified to $V_n = M \times \lambda^n$ with previous approaches seeking to minimize the number of rounds, $n$, needed to copy memory pages. The authors instead ignore $\lambda^n$ and rewrite the total migration time as:

$$T_{mig} \approx \frac{M}{L} \times \frac{1}{1-\lambda} = \frac{M}{L-R}$$

In this new model, the total migration time is the product of the ratio of VM memory size to allocated bandwidth and page dirty rate. The new denominator, $L - R$, is now referred to as the “net transmission rate”. The total net transmission rate of the network is now an objective function, expressed as:

$$\max \sum_{k=1}^{K} (l_k - X_k r_k)$$

Now, instead of minimizing the down time of each VM migration, the objective becomes to maximize the net transmission rate. This has the effect of reducing the overall migration time. The following constraints are placed on the new objective function:

$$\begin{align*}
\sum_{p \in P_k} x(p) &= l_k, k = 1, \ldots, K \\
\sum_{p \in P_k} x(p) &\leq c(e), \forall e \in E
\end{align*}$$

The first constraint ensures that the flows on the path $p$ use all of their allocated bandwidth. The second constraint ensures that no individual link in the path exceeds its residual bandwidth. Because migrations that have not yet started will not have bandwidth allocated to them, the following constraint is added:

$$l_k \leq \beta \times X_k, k = 1, \ldots, K$$

The $\beta$ constant is added as a threshold which the feasible allocated bandwidth cannot exceed. In order to ensure that the indicator variable $X_k$ is a binary indicator of a migration as well as to ensure the flow along the path between the source and target nodes is greater than zero, the following constraints are added, respectively:

$$\begin{align*}
X_k \in \{0, 1\}, k = 1, \ldots, K \\
x(p) \geq 0, p \in P
\end{align*}$$

Based on the observation that migrations that have been started can not be stopped, two additional constraints are added to the linear programming formulation:

$$\begin{align*}
X_k &\geq X_k^0, k = 1, \ldots, K \\
l_k &\geq l_k^0, k = 1, \ldots, K
\end{align*}$$

These constraints ensure that the indicator and bandwidth values for the current migration are at least the same as the previous values, $X_k^0$ and $l_k^0$, respectively. Hence, once started, a migration cannot be stopped and its bandwidth usage does not decrease [16].

The result is a mixed integer programming (MIP) problem which outputs a sequence of VMs to be migrated along with their respective migration bandwidth allocations. The integer linear programming

This work is licensed under a “CC BY-SA 4.0” license.
bandwidth pairs, the problem is further reduced to finding an optimal sequence of VM migration and original MIP even after removing unnecessary variables in the still cannot find a solution in polynomial time, and the page dirty rate value, \( r_k \), are not needed in most cases. The objective function then becomes:

\[
\max \sum_{k=1}^{K} l_k
\]

The constraints are then reduced to:

\[
\begin{align*}
\sum_{p \in P_k} x(p) &= l_k, & k = 1, \ldots, K \\
\sum_{p \in P_e} x(p) &\leq c(e), & \forall e \in E \\
x(p) &\geq 0, & p \in P
\end{align*}
\]

The modified linear programming formulation still cannot find a solution in polynomial time, even after removing unnecessary variables in the original MIP. In order to constrain the problem of finding an optimal sequence of VM migration and bandwidth pairs, the problem is further reduced to the following linear programming formulation:

\[
\max \sum_{p \in P} x(p) \\
\text{s.t.} \quad \sum_{p \in P_e} x(p) \leq c(e), & \forall e \in E \\
x(p) \geq 0, & p \in P
\]

The problem of finding an optimal sequence of VM migration and bandwidth pairs is now a maximum multicommodity flow problem. However, this problem is still not solvable in polynomial time. Instead, a near-optimal solution is obtained in polynomial time using the work of Fleischer et al in [7]. The result is a solution within \( 1 + \epsilon \) of the optimal solution and can be obtained in polynomial time.

### F. FPTA Algorithm

The authors’ FPTA algorithm makes use of the fully polynomial-time approximation scheme (FPTAS) in [7], and is used to solve the maximum multicommodity flow problem that was obtained by iteratively removing unnecessary variables and constraints in the original MIP problem formulation.

The formal algorithm is shown as Algorithm 1 in [16], but a description follows. Since it is a primal-dual algorithm, let \( u(e) \) denote the dual variables of the problem. Then, for all \( e \in E \), \( u(e) \) is the length of the link \( e \). Let \( \text{dist}(p) = \sum_{e \in P} u(e) \) be the length of the path \( p \) between source and destination nodes \( (s_j, d_j) \).

The algorithm begins by taking as input a network \( (G = (V, E)) \), link capacities \( c(e) \) associated with all links in the network \( (e \in E) \), and migration requests \( (s_j, d_j) \). The lengths of all links are initialized to \( \delta \) (i.e. \( u(e) = \delta, \forall e \in E \)), a function of the desired accuracy \( \epsilon \). All flow rates are initialized to zero (i.e. \( x(p) = 0, \forall p \in P \)). The algorithm then enters two loops. The outer first loop bounds the inner second loop by the desired accuracy level and will iterate \( \lceil \log_{1 + \epsilon} \frac{14 + \epsilon}{\delta} \rceil \) times. The second loop augments the flow along the shortest path \( p \) from \( s_j \) to \( d_j \) by adding the capacity of the minimum capacity link in the path. The second loop continues until the length of the path between \( s_j \) and \( d_j \) is between 1 and \( 1 + \epsilon \). The result is that the flow is balanced among all the links in the path between source and destination nodes \( (s_j, d_j) \). A third loop

30 The minimum vertex cover problem seeks to “cover” (or rather, make incident) all the edges in a graph with the minimal number of vertices. Hence, the minimal number of vertices will have at least one edge they’re incident on for all edges in the graph.

31 See the paragraph above regarding ILPs, which are really just a special case of linear programming.

32 The multicommodity flow problem seeks to find a flow assignment that satisfies link capacity constraints, flow conservation constraints (i.e. a flow that enters a node should not change when it leaves that node), and demand constraints (generated by each commodity). Maximizing this just means that the total throughput in the flow network is maximized.

This work is licensed under a “CC BY-SA 4.0” license.
scales each flow by a factor of \( \log_{1+\epsilon} \frac{1+\epsilon}{\delta} \). Thus, according to the work in [7], the final flow is feasible. The authors contribute to this a fourth loop that adds the feasible bandwidth allocation \( l_j \) and migration indicator variable \( I_j \), both of which are provided to the primal MIP problem.

**G. Bound Analysis**

Since FPTA is a primal-dual algorithm with the MIP formulation as the primal problem and the FPTA algorithm as its dual, bounds are computed by first finding a bound for the MIP problem, then finding a bound for the FPTA algorithm. The overall result is that FPTA has a bound of \( (1 - 2\epsilon - \sigma)U \), where \( \epsilon \) is the desired accuracy, \( \sigma \) is a ratio of page dirty rate to net transmission rate, and \( U \) is the optimal result from the primal MIP problem.

The overall complexity of FPTA can be easily computed. FPTA is based on the work of [7], which is an improvement on the work of [8]. The improvements in [7] have a complexity of \( O(\epsilon^{-2}m^2 \log(m)) \) where \( \epsilon \) is the desired accuracy and \( m \) is the number of links in the network. As mentioned previously, an additional loop was contributed to the existing algorithm that adds the feasible bandwidth allocation \( l_j \) and migration indicator variable \( I_j \). The overall complexity is then \( O(\epsilon^{-2}m^2 \log(m) + K) \), where \( K \) represents the additional loop.

**H. Results and Conclusions**

Simulations of FPTA were done by emulating the topology of two data centers. The first is a private data center (PRV1) in the mid-western United States. The second is Google’s B4, a wide-area network that spans a global set of data centers. The PRV1 data center is detailed in [5] and the B4 data center is described in [11]. A brief topology overview of both is provided so the authors’ testing and results have some context.

The PRV1 data center contains 96 network devices and 1,088 servers. No further topology details are provided other than the topology follows the canonical 2-Tier Cisco architecture [5]. Typical data center design best common practices dictate that each server should have at least two connections to its top-of-rack switch (preferably to two distinct top-of-rack switches). With this in mind, there should be at least 2,176 server connections and several hundred additional connections between the top-of-rack layer and the collapsed core/aggregation layer.

The B4 datacenter network is built using custom designed switches which are controlled by Google’s SDN controllers. No information on the number of network devices and servers is provided in [11]. Instead, the focus of the authors’ work is on inter-data center migrations among the 12 B4 data centers which are connected with 19 links. Thus, a node represents a data center instead of an individual server.

Performance is compared between the FPTA algorithm, the optimal MIP formulation, and two additional algorithms. The first additional algorithm was proposed in [9] and [12] and does one-by-one migrations. The second does migrations in groups and was proposed in [4]. The migration time compared to number of migrations, means of memory size, and background traffic are the evaluation criteria.

The simulated network is constrained to 1 GBps and the page dirty rate is set to 100 MBps. The remaining dirty page threshold, \( V_{thd} \), is set to 100 MB and the time spent to restart a VM, \( T_r \), is set to 20 ms. The memory sizes of the VMs range from 1 GB to 10 GB.

For the PRV1 topology and 100 migrations, the performance of the one-by-one algorithm is several times worse than any other tested algorithm, with a migration time of about 100 seconds for every 20 VMs migrated, 100 seconds for every gigabyte of memory, and around 50 additional seconds of migration time for every 100 MB increase in background traffic. The grouping algorithm’s performance stayed close to FPTA’s performance until greater than 50 VMs were migrated, at which point FPTA’s performance was far better. A similar result was observed for memory size and background traffic. This would seem to suggest that FPTA is better suited for a large number of migrations and makes sense intuitively since the objective of FPTA is to maximize bandwidth utilization. FPTA and the MIP formulation showed the same overall performance throughout.

For the B4 topology and 50 migrations, the one-by-one algorithm is omitted from the results due to its extremely poor performance relative to the

---

This work is licensed under a **“CC BY-SA 4.0” license.**
others. As in the PRV1 topology, the grouping algorithm’s performance was worse than FPTA for all the criteria as the number of migrations increased. Interestingly, the same was true between FPTA and the MIP formulation once 20 migrations were exceeded, confirming that the MIP formulation is closer to optimal than FPTA. A separate histogram is presented that shows the difference in computation time between solving the MIP problem using the GLPK solver compared to FPTA’s near-polynomial time bound.

Taken together, the results of PRV1 and B4 show that when the objective is to maximize the net transmission rate of VM migrations, the FPTA algorithm provides significant gains over the grouping algorithm with an acceptable trade off in accuracy from the MIP formulation. The difference in downtime was also significant and averaged a half second.

IV. My Conclusions

Both ORS and FPTA seek to make better use of limited network and node resources. While ORS seeks to embed a virtual network and make ongoing changes to the embedding as other embeddings come and go, FPTA seeks to maximize link utilization for VM migrations, thereby reducing overall migration time. The background material for FPTA included a paper, [22], whose primary focus was on the so-called Virtual Data Center (VDC) embedding problem. The description of the VDC problem bares a striking similarity to the Virtual Network Embedding (VNE) problem which ORS attempts to solve.

ORS has a number of flaws worth mentioning. First, there is no provision for situations where the basic subrequirements exceed the variable subrequirements. Whole time slots are used for basic subrequirements while fractional amounts of time slots are opportunistically shared for variable sub-requirements, suggesting the possibility of a lopsided embedding request. A possible solution to this would be to compare the ratio of basic to variable subrequirements for embeddings and queue them accordingly. Second, batch processing is not supported, though this is mentioned by the authors. Another flaw is the complete lack of load balancing of requests. Residual resources are time slots in links and nodes (bandwidth and CPU, respectively), but availability of a time slot doesn’t preclude that node or link from being completely utilized by other embeddings. Finally, and perhaps most critically, no provision exists for recovering from substrate failures. It is certainly conceivable that a substrate node could fail or a substrate link could be severed, leading to lost revenue or worse, lost data.

FPTA also has some shortcomings worthy of discussion. No mention is made of how residual capacity is calculated for network links. Certainly this is a critical component when the objective is maximum link utilization. Second, a constant, \(\beta\), is attached to the constraints of the MIP problem formulation, but no explanation of how this value is obtained other than “\(\beta\) is a constant large enough so that the maximum feasible bandwidth allocated for each migration cannot exceed it.” Without such an explanation, the results of the entire paper could be in question. Finally, like ORS, no method of recovering from failure is proposed. This is especially important since the paper states that “migrations that have been started cannot be stopped”.

A. Comparisons between Paper 1 and Paper 2

A summary of the algorithms is given in Table 1.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Method</th>
<th>Solves</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORS</td>
<td>Greedy placement of basic subrequirements; probabilistic placement of variable subrequirements based on collisions</td>
<td>VNE problem with varying subrequirements</td>
</tr>
<tr>
<td>FPTA</td>
<td>Primal-dual MIP and ((1 + \epsilon))-approximation algorithm</td>
<td>Minimize migration time by maximizing network utilization</td>
</tr>
</tbody>
</table>

TABLE I
COMPARISON OF ORS AND FPTA.

V. Future Work

Improvements to ORS could include fixing the shortcomings mentioned previously. Ideas for future work are numerous and could include:

- A method for the InP to estimate required substrate resources to service a desired number of VN embeddings
- A study of other possible substrate resources and how to best allocate them

This work is licensed under a “CC BY-SA 4.0” license.
• A method for fragmenting a resource request across a heavily utilized substrate (i.e. break apart variable subrequirements as much as possible once some substrate utilization threshold has been exceeded instead of rejecting the request)
• An exploration of profiling requirements to better capture time-varying resource requirements
• An online method for adding and removing virtual network components to existing embeddings
• How to best choose the \( b, v, p \) values
• Can \( b, v, p \) values change during the life of the embedding?
• How to deal with long-lived or resource-hogging embeddings
• How does the span of a VNE affect its lifetime?
• How to deal with traffic patterns between VN embeddings

FPTA’s shortcomings should be fixed. Future work could include:
• A way to calculate residual bandwidth for the path between \( s_j \) and \( d_j \)
• The SDN controller is superfluous and should be removed. Explore the results of doing so.
• Since FPTA was tested in a wide-area network (Google’s B4 WAN), how might incorporating latency and packet loss measurements affect migrations?
• A study of how the RSVP-TE protocol would decrease migration times by complementing traffic measurements
• A study of how to best choose a \( \epsilon \) value. No mention of this is made in the paper.

REFERENCES

This work is licensed under a “CC BY-SA 4.0” license.