Modeling Service Applications for Optimal Parallel Embedding

Changcheng Huang
Department of Systems and Computer Engineering
Carleton University
Ottawa, Canada
huang@sce.carleton.ca

Jiafeng Zhu
Huawei Technologies Inc.
2330 Central Expy
Santa Clara, USA
jiafeng.zhu@huawei.com

***Abstract* — Leveraging the traditional virtual network concept, some recent research works have proposed the Virtual Data Center (VDC) as an abstraction to capture both bandwidth and compute/storage resource requirements for an application. However a virtual node in a VDC is limited to a virtual machine (VM), which can only be embedded onto a single physical machine. This condition limits the applicability of the VDC abstraction and the potential of deploying parallel computing. In this paper, we propose a new abstraction based on our Application Centric Network Virtualization (ACNV) approach. Specifically, we model a service application offered by a service provider as a virtual network of service function nodes, which closely matches the service provider's view on the architecture of the application. An infrastructure provider that hosts the application decides how to map the virtual network to the substrate network. Different from the VDC abstraction, each virtual node in our proposed abstraction can be split and mapped onto multiple physical machines, which allows the infrastructure provider to provide auto scaling for the application with variable number of physical machines for exploring the full benefits of parallel computing. We also allow multiple virtual nodes to be mapped and colocated in the same physical machine to minimize resource fragmentation and communication overhead. Extensive simulation results show that the proposed ACNV abstraction outperforms existing VDC-like approaches in achieving optimal resource usage.**

Keywords—Architecture; Distributed network; Network topology, Distributed application; Modeling technique, Distributed programming

I. INTRODUCTION

Cloud computing is becoming the world-wide computing paradigm for low-cost computing services. Today's cloud computing is built upon massive datacenters that deploy large number of commodity switches and servers. While these switches and servers drive down the cost for cloud computing, they also pose significant challenges for applications to utilize these resources efficiently without introducing unnecessary overhead. Many programming models such as MapReduce [1] have been developed to achieve large-scale parallel computing with little extra cost. However it remains unsolved for an infrastructure provider (InP) to optimally host multiple service applications within its cloud infrastructure.

Some recent research works have proposed the Virtual Data Center (VDC) [2-5] as an abstraction for the interface between service providers (SPs) and InPs. The VDC concept originates from the traditional network virtualization that is focused on embedding multiple virtual networks (VNs) onto a shared network substrate [6-7]. The VDC abstraction models an application as a virtual network. It is assumed that an SP needs to convert its application into a VDC and presents this VDC to an InP as a requirement. The InP then tries to embed this VDC with the given virtual nodes and topology onto its infrastructure through an optimization process. Similar to the traditional studies on VNs [8-12], each virtual node in VDC is considered as a virtual machine (VM) that can only be mapped onto a single physical machine. This condition limits the applicability of the VDC abstraction and the potential of parallel computing. Because an SP does not have the global knowledge of all applications offered by different SPs sharing the same infrastructure, it is difficult for an SP to decide how many VMs an application requires and how much resource each VM should have. On the other hand, an InP cannot optimize its resource usage by changing number of VMs for each application and the size of each VM although it does have the global view of all SPs.

A. Cloud Services from IaaS to PaaS

The VM as a service is mainly adopted in the Infrastructure as a Service (IaaS) class. Although IaaS is a popular service class, the interest in the so-called Platform as a Service (PaaS) class is growing very fast. A cloud platform offers an environment on which developers create and deploy applications and do not necessarily need to know how many processors or how much memory the applications will be using as long as their service performances are satisfied.

Amazon Web Services (AWS) [13] is one of the major players in the cloud computing market. It serves as a good example for the migration from IaaS to PaaS. It pioneered the introduction of IaaS clouds in 2006. However, it advocates PaaS aggressively in recent years [14]. It is now considered as offering PaaS like services with the option of IaaS-like control in some cases [15]. A cloud-based application offered by a SP typically consists of multiple components with each component performing certain functions. The output of a component can be routed to other components through messaging queues that

help decouple different components and enable asynchronous processing [14]. Bundling a component with operating system and associated configuration can create an Amazon machine image (AMI). Instances (or VMs) can be instantiated from AMIs as needed and run on one or multiple physical machines. Elasticity can be achieved by combining the CloudWatch, Auto Scaling, and Elastic Load Balancing features, which allow the number of instances to scale up and down automatically based on a set of customizable rules, and traffic to be distributed across available instances.

B. System Description

With PaaS becoming popular, we need a new abstraction that better characterizes the interface between SPs and InPs. In this paper, we propose a new interface abstraction based on our Application Centric Network Virtualization (ACNV) approach. Specifically, we model an application as a virtual network in which a virtual node represents a functional component of the application. As we can see from last paragraph, this virtual network closely matches the view and implementation as seen by the SP. The InP can embed this virtual network onto its substrate network through an optimization process. In contrast to the VDC abstraction, our abstraction allows a virtual node to be mapped onto multiple substrate nodes to help realize load balancing and achieve parallel computing. We also allow multiple virtual nodes to be mapped and colocated in one physical machines. This kind of many-to-many mapping enables full flexibility to maximize the efficiency of resource usage.

Furthermore, we propose to classify nodes into three categories based on their roles in a network for both virtual nodes and substrate nodes: transit nodes that relay traffic, but do not originate/sink traffic; stub nodes that originate and sink traffic, but do not relay traffic; and hybrid nodes that do both. Because a hybrid node can be decomposed into a transit node attached with a stub node as shown in Fig.1, we will focus on the first two types of nodes in this paper.

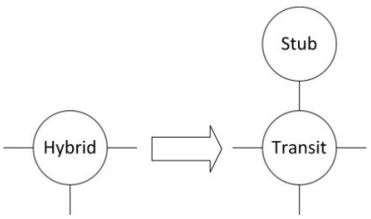

Fig. 1. Illustration of decomposing a hybird node into a transit node attached with a stub node.

A substrate transit node (STN) such as a switch is typically limited by its transit bandwidth capacities rather than by its compute or storage resources. An SP can define some virtual transit nodes (VTNs) for an application that can help elaborate its communication requirements for the application. This issue has been studied in detail in [3] under two special virtual topologies. In this paper, we propose a generic formulation that can map arbitrary topologies with any given traffic patterns. We allow a VTN to be split onto multiple STNs selected by the SP as well as multiple VTNs to colocate in the same substrate node. These different options provide great flexibility to accommodate different traffic patterns and therefore are helpful in mitigating various congestion problems [3].

Substrate stub nodes (SSN) are limited by their compute and storage resources. Different from existing VN models, we also allow a virtual stub node (VSN) to be split onto multiple SSNs in addition to colocating multiple VSNs at the same SSN.

With the above extensions, we can model various applications that are supported by today's datacenters. For example, MapReduce is a distributed programming framework that is widely used in Today's datacenters [1]. It includes a mapping stage and a reducing stage. In the mapping stage, a large number of Mapper tasks run in parallel to perform filtering and sorting. The outputs of the mapping stage are shuffled to a large number of Reducer tasks that perform summary operations in parallel. Using our modeling approach, a MapReduce application can be modeled as a VSN (Node 1) that represents the Mapper function, one VTN (Node 2) that represents the shuffling process, and another VSN (Node 3) that represents the Reducer function as shown in Fig. 2. The three nodes are connected into a linear topology that greatly simplifies an SP's view on the application. The SP only needs to specify CPU load requirements for Nodes 1 and 3 and the bandwidth requirements for the two virtual links and Node 2. The InP will decide how to map this virtual network to its substrate network. With node splitting, Nodes 1 and 3 can be mapped onto large clusters of SSNs respectively to support parallel computing while Node 2 can be mapped onto a cluster of STNs through the optimization process discussed later as illustrated in Fig.2. The two virtual links will be mapped onto a large number of substrate paths to utilize the rich connectivity within a datacenter network.

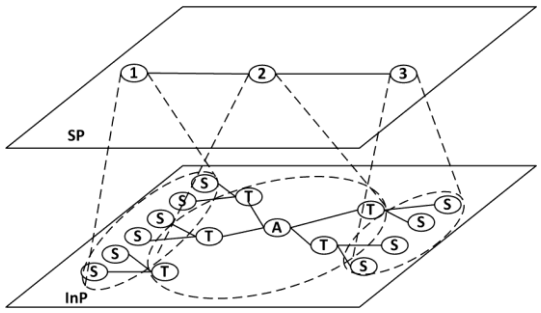

Fig. 2. Modeling MapReduce computation with virtual nodes. S stands for servers; T stands for Top of Rack switch; A stands for aggregation switch.

Another area our new abstraction can be applied to is Network Function Virtualization (NFV). Traditionally a service function chain (SFC) consists of a set of dedicated network service boxes such as firewall, load balancers, and application delivery controllers that are concatenated together to support a specific application. With a new service request, new devices must be installed and interconnected in certain order. This can be a very complex, time-consuming, and error-prone process, requiring careful planning of topology changes and network outages and incurring high OPEX. This situation is exacerbated when a tenant requires different service sequences for different traffic flows or when multiple tenants share the same infrastructure network. NFV is a new concept

built upon network virtualization. It involves the implementation of network service functions in software that can run on a range of industry standard high volume servers, switches, and storage. To handle large volume of traffic, each service function can be mapped to multiple physical machines with auto scaling to achieve parallel processing and load balancing. It is easy to see that our abstraction can be applied to NFV based SFCs with each service function modeled as a virtual node.

The remaining part of this paper is organized as follows: Section II will describe our formulation process. Numerical results will be presented in Section III. Section IV will briefly review related literature and highlight the key contributions of this paper. Section V will conclude this paper.

II. Network Model and Problem Formulation

In this section, we first describe our network model that can capture all mapping scenarios. We then formulate the optimization process as a Linear Programming (LP) problem. Instead of limiting our discussion to specific applications or datacenter topologies, we will consider scenarios as general as possible so that our approach can be applied to a wide range of applications. Fig. 3 shows an example with an arbitrary virtual network being mapped to an arbitrary substrate network. We will use this example to illustrate the formulation process.

A. Substrate Network

A substrate network is modeled as a weighted directed graph and denoted as $G^S = (N^S, L^S)$, where N^S is the set of substrate nodes and L^S is the set of substrate links. Each node $n^S \in N^S$ has a location $a(n^S)$.

Each substrate link $l^S \in L^S$ between two substrate nodes u^S and v^S has the bandwidth capacity value $B(l^S)$.

Different from existing approaches [8-12], we divide N^S into two subsets: the set of all SSNs denoted by N^{SS} such as Nodes *A*, *B*, *G*, *I*, and *J* in Fig. 3 and the set of all STNs denoted by N^{ST} such as Nodes *C*, *D*, *E*, *F*, and *H* in Fig. 3. $N^S = N^{SS} \cup N^{ST}$ and $N^{SS} \cap N^{ST} = \emptyset$ because a substrate node is either a stub node or a transit node but not both. Each $n^{SS} \in N^{SS}$ has a CPU capacity value $C(n^{SS})$. To simplify notations, we do not consider memory and storage resources in this paper. Our approach can be generalized to include those resources easily. Interested readers can see the treatments in [4-5] as an example. We assume an SSN will not carry transit traffic, which is the typical case for today's datacenters. But we assume there are unlimited bandwidths among colocated VSNs enabled by Hypervisor because the bandwidths within a substrate server are rarely exhausted due to the short distance and large amount of memory available.

Most of today's switches or routers are non-blocking internally (i.e. the internal switch fabric speed is much faster than each output port speed). Traffic can only be blocked by limited bandwidths of output ports as defined earlier by the link bandwidths. Therefore we assume that STNs have unlimited internal bandwidths and limited port bandwidths. We will focus on how VTNs can satisfy their transit bandwidth requirements through mapping onto one/multiple STN(s). Our approach in this paper can be generalized to the cases in which the internal bandwidths of STNs are limited.

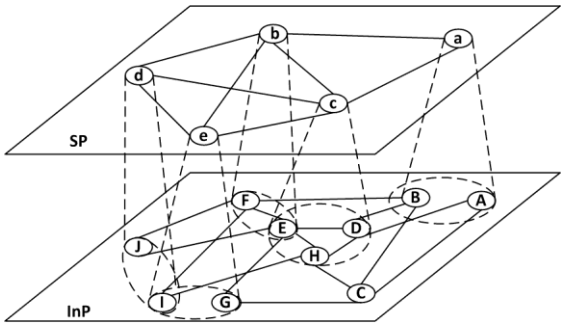

Fig. 3. An example of a virtual network and its associated substrate network.

B. VN Request

We model each VN as a weighted directed graph, denoted by $G^V(t,h) = (N^V, L^V, t, h)$, where N^V is the set of virtual nodes, L^V is the set of virtual links, t is the arrival time of the VN request and h is the duration of the VN request. The set of all virtual networks hosted by the substrate network at time t is denoted by $G^A(t) = \{G^V(s,h) | s \le t \le s+h\}$. Whenever the context is clear, we simply write $G^V(t,h)$ as G^V. We denote the ith virtual link of a virtual network as $l_i^V \equiv (s_i^V, d_i^V)$, where s_i^V and d_i^V are the corresponding start and end virtual nodes. Each virtual link has an associated bandwidth requirement $B(l_i^V)$.

Similarly, we divide N^V into two subsets: the set of all VSNs denoted by N^{VS} such as Nodes *a*, *d*, and *e* in Fig. 3 and the set of all VTNs denoted by N^{VT} such as Nodes *b* and *c* in Fig. 3. An SP may prefer mapping a virtual node onto some specific physical nodes or dictating some virtual nodes to be mapped onto different physical nodes. To facilitate this kind of requirements, it is assumed that each virtual node $n^V \in N^V$ has a location $a(n^V)$ and a mapping radius $D(n^V)$. Let $d(a(n^V), a(n^S))$ denote the distance between the virtual node n^V and an arbitrary substrate node n^S (measured in any metrics as defined by the SP such as hop count or delay in a datacenter network). If $d(a(n^V), a(n^S)) \le D(n^V)$ and n^V is the same type of node as n^S (i.e. they are either both stub nodes or transit nodes), n^S will be called a candidate node for the virtual node n^V to be mapped onto. By defining candidate node, we allow an SP to specify some rules for substrate node selection. It should be noted that $D(n^V)$ does not need to be a physical distance. For example, it can be a hop count. Then by setting $D(n^V) = 2$, we can limit the candidate servers to those under the same Top of Rack (ToR) switch. This will maximize performance while lowering reliability. If an SP wants to maximize reliability, distance can be defined as a measure of zones. Then the optimization process can lead to selecting servers located in different zones.

We do not allow VTNs to be mapped onto SSNs and vice versa because they have different types of resources. The set of all candidate nodes for n^V is denoted as $N^C(n^V)$. The substrate link that connects two candidate nodes of a virtual node is called internal substrate link. For example, in Fig. 3, the candidate set of the VSN *e* includes Nodes *I* and *G*; the

candidate set of the VTN c includes Nodes D, E, and H. Link (E, H) is an internal link. It is interesting to note that, Nodes e and d can be partially colocated at Node I, similarly Nodes b and c at E.

The set of all VSNs hosted by an SSN $u^{SS} \in N^{SS}$ is denoted as $H^S(u^{SS})$ (e.g. Nodes d and e hosted by Node I) and the set of all VTNs hosted by an STN is denoted as $H^T(u^{ST}), u^{ST} \in N^{ST}$(e.g. Nodes b and c hosted by Node E). Each VSN $n^{VS} \in N^{VS}$ requires a CPU capacity value $C(n^{VS})$. We assume that $C(n^{VS})$ can be arbitrarily split and mapped onto any number of SSNs selected by an SP. This assumption is valid because new computing models such as MapReduce can distribute a computation to hundreds or thousands of machines as they are being used in today's datacenters. The granularity is small enough to be considered as real number.

For each $n^{VT} \in N^{VT}$, let $L(n^{VT})$ denote the set of all its adjacent virtual links. Different from VSNs, we assume each VTN has certain transit requirement rather than CPU load. A VTN typically has multiple ports. The transit capacity between each pair of virtual links incident to the virtual node must be specified in order to ensure traffic can be carried through the virtual node without any congestion. This means the SP must specify a traffic pattern within each VTN.

We assume each pair of adjacent virtual links require a transit bandwidth value $B(l^T(l_i^V, l_j^V))$, where $l^T(l_i^V, l_j^V)$, $l_i^V, l_j^V \in L(n^{VT})$, identifies a virtual transit link, and l_i^V is the incoming virtual link and l_j^V is the outgoing virtual link. A virtual transit link can only be mapped to a set of paths in the substrate graph made up by the internal substrate links of the VTN. The set of all virtual transit links of an n^{VT} is denoted by $L^T(n^{VT})$. Clearly $B(l^T(l_i^V, l_j^V))$ depends on $B(l_i^V)$ and $B(l_j^V)$, as well as all other virtual transit links. This will make our formulation results very different from existing research studies that do not consider traffic pattern within a virtual node at all. We will discuss how to select $B(l^T(l_i^V, l_j^V))$ in detail later.

C. VN Mapping

In this paper, we consider the online network virtualization problem, which is more realistic in real world. We assume VN requests arrive dynamically. For each VN request, the InP has to decide whether it has enough resources to host the requested VN, assuming existing VNs will not be reallocated. If the request is acceptable, the InP has to map the VN request to the substrate network. We consider the cases that allow both node colocation and node splitting. These scenarios are certainly more challenging than the traditional setting that only considers colocation at most. We address the challenge by treating VSNs and VTNs differently. We first need to know how many resources are available at any time, upon which we can decide whether a new VN request can be granted.

For an n^{VS}, we denote the CPU capacity value assigned to an arbitrary candidate node $n^{SS} \in N^C(n^{VS})$ as $C(n^{VS}, n^{SS}) = x_{mw}C(n^{VS})$, where $m = n^{VS}, w = n^{SS}, 0 \le x_{mw} \le 1$. For $n^{SS} \notin N^C(n^{VS})$, we set $C(n^{VS}, n^{SS}) = 0$. We have

$$\sum_{w \in N^C(m)} x_{mw} = 1, \forall m \in N^{VS} \quad (1)$$

The residual CPU capacity of an SSN at any time t can then be defined as following

$$R_N(n^{SS}, t) = C(n^{SS}) - \sum_{n^{VS} \in N^{VS}, G^V \in G^A(t)} C(n^{VS}, n^{SS}) \quad (2)$$

A virtual link is mapped onto multiple paths in the substrate network. The bandwidth value for a virtual link $l^V \in L^V$ assigned to a substrate link $l^S \in L^S$ is denoted by $B(l^V, l^S)$.

For $n^{VT} \in N^{VT}$, its transit bandwidth requirement for each virtual transit link is also mapped onto multiple paths in the substrate network. We denote the bandwidth value of a virtual transit link $l^T(n^{VT})$ assigned to an arbitrary substrate link l^S as $B(l^T, l^S)$.

We can calculate the remaining bandwidth of a substrate link as following:

$$R_l(l^S, t) = B(l^S) - \sum_{l^T \in L^T(n^{VT}), G^V \in G^A(t)} B(l^T, l^S) - \sum_{l^V \in L^V, G^V \in G^A(t)} B(l^V, l^S) \quad (3)$$

D. Augmented Graph

Similar to the approach adopted in [10], we also create an augmented substrate graph $G^{S'} = (N^{S'}, L^{S'})$ for the graph $G^S = (N^S, L^S)$. For each $n^V \in N^V$, a corresponding meta-node $\mu(n^V)$ is created. Each candidate substrate node of $n^V \in N^V$ is connected to the meta-node through a bidirectional meta-link with infinite bandwidth. We set $N^{S'} = N^S \cup \{\mu(n^V) | n^V \in N^V\}$ and $L^{S'} = L^S \cup \{(\mu(n^V), n^S) | n^V \in N^V, n^S \in N^C(n^V)\}$, which form $G^{S'} = (N^{S'}, L^{S'})$. The set of all meta-links incident to a meta-node $\mu(n^V)$ is denoted by $L^{\mu(n^V)}$. If s_i^V and d_i^V are the start and end nodes of a virtual link, $\underline{s_i} = \mu(s_i^V)$ and $\underline{d_i} = \mu(d_i^V)$ will denote the corresponding meta-nodes. The augmented graph for the example in Fig. 3 is shown in Fig. 4.

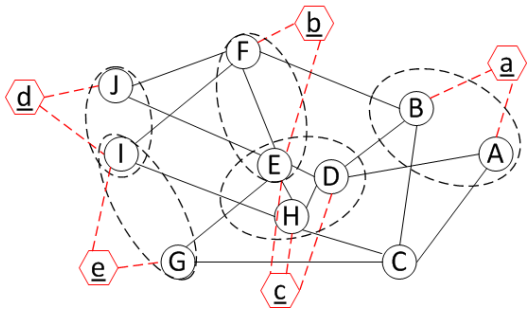

Fig. 4. The augmented graph for the example shown in Fig. 3.

E. Virtual Stub Node Mapping

With all existing studies [2-12], the bandwidth for a virtual link is typically given independently from the CPU capacity

requirements of its associated VSNs. This is fine if a virtual node is mapped onto a single substrate node. In this paper, we allow a virtual node to be mapped onto multiple substrate nodes. A new challenge will emerge with this new model. The CPU requirement of the virtual node is assigned to multiple candidate substrate nodes while the bandwidth requirement of its any adjacent virtual link is mapped to multiple substrate links associated with different candidate nodes. There is an issue on whether the traffic originated/terminated from/at a substrate node for a virtual link should be somehow correlated to the CPU capacity assigned to the substrate node.

An easy option is to keep bandwidth assignments independent from CPU assignments as the way existing research studies have done for non-splitting node mapping [2-12]. However this may oversimplify the problem because it is unlikely that a node providing lots of CPU power will generate very little communication traffic. To this end, a more practical assumption is that the traffic originated/terminated from/at an SSN for a virtual link is proportional to the CPU power assigned to the substrate node for the associated virtual node. This approach keeps linear relationships among all variables while making allocated bandwidth for each substrate node correlated to the CPU load assigned to the node. We will follow this approach in this paper. More details can be found in remarks about Eqs. (13), (15), (16), and (20) later. An example on how the bandwidth requirement of virtual link (a, c) in Fig. 3 is split is shown in Fig. 5.

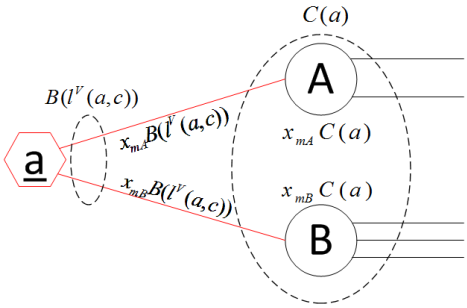

Fig. 5. An example to show how the bandwidth requirement of virtual link (a, c) in Fig. 3 is split.

F. Traffic Pattern for Virtual Transit Node

As we mentioned earlier, an SP needs to specify a bandwidth requirement for each virtual transit link within a VTN. This requirement will be mapped onto the internal substrate links of the VTN in the substrate network. An internal substrate link is a link that connects two candidate nodes of the same VTN (e.g. link (D, E) in Fig.4).

It is a challenging task for an SP to specify the bandwidth requirement for each virtual transit link because virtual transit links of a VTN are interdependent. This challenge is best illustrated with Fig. 6, in which a virtual node with four input virtual links and four output virtual links is shown together with some traffic patterns. In Fig. 6(a), the traffic pattern is uniform in the sense that no more than one virtual link is sending traffic to the same output virtual link at the same time. Fig. 6(b) shows a more difficult situation where all input virtual links are trying to send traffic to one output virtual link with their full capacities at the same time (e.g. an incast flow pattern). In reality, traffic patterns are rarely like the two extreme cases we have just shown.

Other than interdependency, the bandwidth requirement of a virtual transit link also depends on the bandwidths of input and output virtual links. In this paper, similar to prior studies, we assume the bandwidths of all virtual links are given by the SP. Clearly, there is no point to request for the bandwidth of a virtual transit link much larger than the bandwidths of the corresponding input and output virtual links. The impact of traffic patterns are also discussed in [3] under two special virtual topologies. Our approach here is more general.

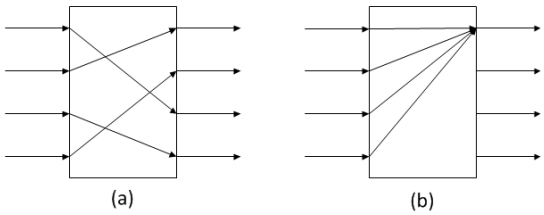

Fig. 6. A virtual node with four input virtual links and four output virtual links under two different traffic patterns.

To characterize traffic patterns, we define the term *distribution ratio* denoted by $\alpha_{ij}(n^{VT})$ as such that $\alpha_{ij}(n^{VT})$ fraction of the traffic coming in from the *i*th input virtual link will be carried by the virtual transit link to *j*th output virtual link. Wherever the context is clear, we will simply drop the part in brackets to simplify our notation. Clearly we have

$$\sum_{j|l_j^V \in L(n^{VT}), j \neq i} \alpha_{ij} = 1, \forall i | l_i^V \in L(n^{VT}) \qquad (4)$$

As an example, we use Fig. 7 to illustrate the traffic flows for VTN c in Fig. 3 with virtual link (a, c) as the incoming link. The traffic coming from Node a is split into three streams leading to Nodes b, d, and e respectively with the ratios as shown in Fig. 7.

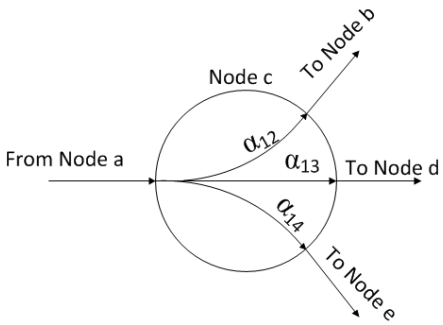

Fig. 7. An illustration on how traffic from a virtual link (a, c) is split into three output streams within VTN c.

It is easy to see the following conditions must be satisfied in order to make a VTN non-blocking.

$$B\left(l^T\left(l_i^V, l_j^V\right)\right) \geq \alpha_{ij} B(l_i^V), \forall j \neq i | l_i^V, l_j^V \in L(n^{VT}) \qquad (5)$$

$$B(l_j^V) \geq \sum_{i|l_i^V \in L(N^{VT}), i \neq j} \alpha_{ij} B(l_i^V), \forall j | l_j^V \in L(n^{VT}) \quad (6)$$

Eq. (5) makes sure that the virtual node is non-blocking internally while Eq. (6) makes sure the output ports are non-blocking.

Similarly, we can define $\beta_{ij}(n^{VT})$ as the fraction of output virtual link j's traffic received through a virtual transit link from input virtual link i. We will also drop the part in brackets wherever the context is clear. Therefore we have

$$\beta_{ij} = \frac{\alpha_{ij} B(l_i^V)}{\sum_{i|l_i^V \in L(N^{VT}), i \neq j} \alpha_{ij} B(l_i^V)}, \forall i \neq j | l_i^V, l_j^V \in L(n^{VT}) \quad (7)$$

Clearly,

$$\sum_{i|l_i^V \in L(n^T), i \neq j} \beta_{ij} = 1, \forall j | l_j^V \in L(n^{VT}) \quad (8)$$

Then, Eq. (6) can be simplified to the following one.

$$\beta_{ij} B(l_j^V) \geq \alpha_{ij} B(l_i^V), \forall i \neq j | l_i^V, l_j^V \in L(n^{VT}) \quad (9)$$

By combining Eqs. (5) and (9) , we have

$$\alpha_{ij} B(l_i^V) \leq B\left(l^T(l_i^V, l_j^V)\right) \leq \beta_{ij} B(l_j^V), \quad \forall i \neq j | l_i^V, l_j^V \in L(n^{VT}) \quad (10)$$

It is wasteful to set $B\left(l^T(l_i^V, l_j^V)\right)$ larger than $\beta_{ij} B(l_j^V)$.

The conditions defined in Eq. (10) will help an SP reduce resource waste and prevent congestion. In this paper, we assume that an SP will provide a traffic pattern for each VTN, which satisfies Eq. (10). The SP can do so by analyzing its user traffic history or through online measurements. This process is very similar to the process that an InP builds and dimensions its physical networks. Our focus in this paper will be on mapping the traffic patterns specified by an SP onto the substrate network.

G. Objective

An InP typically wants to maximize its revenue while minimize its cost. However, revenue depends on many marketing factors which are subject to change. In this paper, we focus on minimizing the cost of accepting a new VN request as other works [8-12]. The cost of embedding a VN request is defined as the sum of the costs of all substrate resources allocated to the VN.

H. Linear Programming Formulation

Upon receiving a new VN request, an InP will first calculate the remaining resources using Eqs. (2) and (3). The InP will then try to map the VN request onto the remaining resources with the objective that will minimize the overall cost while satisfying the requirements of the VN. To this end, we provide two sets of cost weights so that different resources can be combined into a cost function that can be minimized. Specifically, A_{uv} is the cost weight assigned to a substrate link (u, v). The cost of the link is proportional to its normalized bandwidth usage with a maximum value A_{uv} when its total bandwidth is fully allocated. Similarly, B_w is the cost weight assigned to a substrate node w. These weights can stay unchanged for all VN requests or change from one VN request to another. The InP can manipulate the weights for adjusting the significance of each specific resource as needed. For example, if some physical links have higher costs due to distances or other physical constraints, the InP may set the weights for those links higher so that they have less chance to be selected. If the InP cannot find a mapping that satisfies the VN request, the request will be rejected.

Because we assume that a virtual node can be divided arbitrarily and mapped onto a large cluster of substrate nodes as we argued earlier, we can formulate our embedding process as a two-level Multi-Commodity Flow (MCF) problem. The first-level commodities are the virtual links with their corresponding bandwidths as their demands, where each demand can be routed through multiple paths in the substrate network.

The second-level commodities are virtual transit links of each VTN. We set the demand of each virtual transit link to satisfy Eq. (10), which is routed through internal substrate links of a VTN via one or multiple paths. Due to the dependency between first-level and second-level commodities, the flows of second-level commodities must match the flows of corresponding first-level commodities as defined by Eq. (10).

In this paper we assume a virtual transit link can be mapped onto any paths routed through internal substrate links of a VTN. However in real networks, various physical, protocol, or administrative limitations may be imposed when the traffic of a virtual transit link is divided and distributed over different substrate links. These limitations can be formulated as extra constraints during the optimization process.

Variables:

- f_{uv}^i: A variable denoting the total amount of flow from u to v on a link $l^{S'}(u, v) \in L^S \cup L^{\mu(s_i^V)} \cup L^{\mu(d_i^V)}$ for the i th virtual link $l_i^V(s_i^V, d_i^V)$, where u and v are two neighboring substrate nodes. This variable describes how the bandwidth of a virtual link i is mapped onto a substrate link (u, v).
- f_{uv}^{ijk}: A variable denoting the total amount of flow from u to v, where $u, v \in N^C(k), k \in N^{VT}$, for the virtual transit link that connects incoming virtual link $l_i^V \in L(k)$ to an outgoing virtual link $l_j^V \in L(k)$. This variable describes how the bandwidth of a virtual transit link $l^T(l_i^V, l_j^V)$ is mapped onto an internal substrate link (u, v).
- x_{mw}: A variable denoting the percentage of the CPU capacity weight value of the VSN $m \in N^{VS}$ assigned to a candidate node $w \in N^C(m)$.

Objective:

Minimize

$$\sum_{u,v\in N^S}\frac{A_{uv}}{R_l(u,v)+\delta}\sum_{i|l^i\in L^V} f_{uv}^i + \sum_{k\in N^{VT},i,j|l_i^V,l_j^V\in L(k)}\ \sum_{u,v\in N^C(k)}\frac{A_{uv}f_{uv}^{ijk}}{R_l(u,v)+\delta} + \sum_{m\in N^{VS}}\sum_{w\in N^C(m)}\frac{B_w x_{mw}C(m)}{R_N(w)+\delta} \quad (11)$$

Constraints:

-Capacity Constraints:

$$\sum_{i\in L^V} f_{uv}^i + \sum_{k\in H^T(u)\cap H^T(v),i,j|l_i^V,l_j^V\in L(k)} f_{uv}^{ijk} \le R_l(u,v), \quad \forall u,v\in N^S \quad (12)$$

$$\sum_{m\in H^{SS}(w)} x_{mw}C(m) \le R_N(w), \quad \forall w\in N^S \quad (13)$$

-Flow Constraints:

$$\sum_{v\in N^{ST}\cup\{\underline{d_i}\}\cup N^C(d_i^V)} f_{uv}^i - \sum_{v\in N^{ST}\cup\{\underline{s_i}\}\cup N^C(s_i^V)} f_{vu}^i = 0, \quad \forall i\in L^V, \forall u\in N^{ST}\cup N^C(d_i^V)\cup N^C(s_i^V) \quad (14)$$

$$f_{\underline{s_i}u}^i = x_{s_i^V u}B(l_i^V), \quad \forall i\in L^V,\ \forall s_i^V\in N^{VS}, \quad \forall u\in N^C(s_i^V) \quad (15)$$

$$f_{u\underline{d_i}}^i = x_{d_i^V u}B(l_i^V), \quad \forall i\in L^V,\ \forall d_i^V\in N^{VS}, \quad \forall u\in N^C(d_i^V) \quad (16)$$

$$\sum_{u\in N^C(s_i^V)} f_{\underline{s_i}u}^i = B(l_i^V), \forall i\in L^V, \forall s_i^V\in N^{VT} \quad (17)$$

$$\sum_{u\in N^C(d_i^V)} f_{u\underline{d_i}}^i = B(l_i^V), \forall i\in L^V, \forall d_i^V\in N^{VT} \quad (18)$$

$$\beta_{ij}f_{\underline{k}u}^j + \sum_{v\in N^C(k)} f_{uv}^{ijk} - \alpha_{ij}f_{u\underline{k}}^i - \sum_{v\in N^C(k)} f_{vu}^{ijk} \ge 0, \quad \forall u\in N^C(k), \underline{k}=\mu(k), k\in N^{VT}, l_i^V, l_j^V\in L(k) \quad (19)$$

-Meta Constraints

$$\sum_{w\in N^C(m)} x_{mw} = 1, \forall m\in N^{VS} \quad (20)$$

-Domain Constraints

$$f_{uv}^i \ge 0, \forall u\in N^S\cup\underline{S_i}, \forall v\in N^S\cup\underline{d_i} \quad (21)$$

$$f_{uv}^{ijk}\ge 0, \forall u,v\in N^C(k), \forall i,j|l_i^V,l_j^V\in L(k), \forall k\in N^T \quad (22)$$

$$x_{mw}\ge 0, \forall m\in N^{VS}, \forall w\in N^C(m) \quad (23)$$

Remarks:

- Function (11) is the objective function that tries to minimize the total cost while balancing the loads across different substrate links and substrate nodes. For the load of each link, we count both the bandwidths allocated for virtual links and the bandwidths allocated for virtual transit links. δ is a small positive number used to avoid dividing by zero. The load of each link or node is normalized by its residual capacity. Therefore the links or nodes which have less residual capacities tend to have higher costs. This will encourage balanced loads across the whole network. A_{uv} and B_w are cost weights used by an InP to control the significance of the corresponding link or node. $A_{uv}\ge 0$ and $B_w\ge 0$. $R_N(w)$ and $R_l(u,v)$ can be calculated using Eqs. (2) and (3) respectively.
- Constraints (12) and (13) are the link and node capacity bounds.
- Constraint (14) is the flow conservation conditions for virtual link commodities. Each flow can only traverse STNs and the candidate nodes of the two ending virtual nodes of a virtual link to avoid using an SSN as transit node. It should be noted that constraint (14) allows source and destination virtual nodes of a virtual link mapped onto the same substrate node. Take Node *E* in Fig. 3 as an example. Node *E* can serve as both source and destination of the virtual link (*b*, *c*).
- Constraints (15) and (16) ensure the traffic originated from or sunk at a substrate node for a virtual link is proportional to the allocated CPU capacity from the associated VSN to the substrate node as we argued earlier.
- Constraints (17) and (18) ensure the bandwidth requirement for each virtual link where the corresponding virtual nodes are VTNs is satisfied.
- Constraint (19) is a variation of flow conservation conditions for each virtual transit link as second-level commodity. It makes sure first-level flow amounts (i.e. $f_{\underline{k}u}^j$ and $f_{u\underline{k}}^i$) and second-level flow amounts (i.e. f_{uv}^{ijk} and f_{vu}^{ijk}) satisfy Eq. (10) at flow level. Specifically, $\beta_{ij}f_{\underline{k}u}^j$ is the amount of the traffic carried away by *j*th virtual link from Node *u* for the virtual transit link $l^T(l_i^V, l_j^V)$. f_{uv}^{ijk} is the amount of traffic carried away by the internal link (*u*, *v*) from Node *u* for the virtual transit link $l^T(l_i^V, l_j^V)$. $\alpha_{ij}f_{u\underline{k}}^i$ is the amount of traffic carried into Node *u* from *i*th virtual link for the virtual transit link $l^T(l_i^V, l_j^V)$. f_{vu}^{ijk} is the amount of traffic carried into Node *u* by the internal link (*v*, *u*) for the virtual transit link $l^T(l_i^V, l_j^V)$.
- Constraint (20) is the normalization requirement for CPU capacity assignments. It allows a VSN to be mapped onto multiple candidate SSNs.
- Constraints (21), (22) and (23) denote the real domain constraints.

III. Numerical Results

In this section, we present our numerical results. Because our network model is significantly different from all existing approaches, it is difficult to make a fair comparison with the existing ones directly. Therefore we take a two-step approach. In the first step, we assume all substrate and virtual nodes to be homogeneous nodes where substrate nodes have limited CPU resources and unlimited transit bandwidths while virtual nodes only have CPU requirements. Under this context, the differences between our model and the existing ones are limited to different mapping options. Our attention is then focused on the impacts of node splitting and colocation in comparison with the existing ones. In the second step, we will include node types as evaluation factors. Because it is hard to make a fair comparison under this situation, we will focus on the performances of our VN model with varying parameters without comparing with existing approaches.

A. Homogeneous Node Model

In this subsection, we consider the case in which all nodes, both substrate and virtual, are homogeneous. All substrate nodes have limited CPU capacities and unlimited transit bandwidths. This assumption is similar to most of existing approaches.

There are quite a few papers on NV in recent years. It is hard to choose which one to compare with. We decided to choose [10] as the competitive one for several reasons. First of all, the paper is well accepted in academic world and widely cited; Second, the paper provides detail information about its simulation process. This allows us to reproduce their results with excellent accuracy. Third, the paper uses random generated topologies that are consistent with our general approach described in the last section. We do not want to limit our simulation to specific topology that cannot demonstrate the generality of our approach.

In specific, we choose D-ViNE (Deterministic node mapping with MCF link mapping) in [10] as our reference model. The reason is that D-ViNE shows the best performance among all algorithms discussed in [10]. As most of existing studies [2-12] on NV, D-ViNE only allows one-to-one mapping. Node splitting and node colocation are not allowed. Although the study in [12] has considered colocation, the available information is limited, making it hard to reproduce the results. We denote our scheme as ACNV (Application-Centric Network Virtualization) for convenience of description. We expect that our ACNV approach will outperform D-ViNE significantly because ACNV allows many-to-many mapping that can take advantage of node splitting and node collocation for reducing resource fragmentation and communication overhead.

1) Simulation Setup

We have implemented a discrete event simulator that captures the dynamic processes of service request arrivals as well as online network optimization and resource allocations. The optimization tool we used is glpk [19].

A substrate network was generated with 50 nodes in 25×25 grids using GT-ITM tool [20]. The links that connect each pair of substrate nodes were randomly generated using Waxman 2 model with α=0.5 and β=0.2 [20]. There were 123 links in total.

As discussed in the last section, both allocated bandwidth and CPU resources are normalized by their corresponding link or node residual capacities in the cost function (11). Therefore, the absolute values of link and CPU capacities do not matter in terms of the optimization results. It is their relative values that make differences. In our setup, we try to see the impacts of both link and node capacities. Therefore, we chose $A_{uv} = B_w = 1$, $\forall u, v \in N^S, w \in N^{ST}$. The CPU resources of the substrate nodes and bandwidth resources of the substrate links were generated randomly with real numbers uniformly distributed in the same range between 50 and 100. In this way, we did not specifically try to favor one resource over the other.

VN requests arrived in Poisson processes with varying VN arrival rates from average four VNs per 100 time units to eight VNs per 100 time units. Each VN request had an exponentially distributed lifetime with a mean of 1000 time units. In each VN request, the number of virtual nodes was randomly generated by a uniform distribution between 2 and 10. Links between each pair of virtual nodes were also generated using Waxman 2 model with α=0.5 and β=0.2. The bandwidth requirements of virtual links were uniformly distributed between 0 and 50 while the CPU requirements of virtual nodes were uniformly distributed between 0 and 20 which is small because D-ViNE does not support node splitting. Virtual nodes were randomly located on 25×25 grids. The results will highly favor our ACNV. Each simulation was run for 50000 time units that are long enough (50 times longer than the average lifetime of a virtual network) to have sufficient number of independent samples so that statistical errors are negligible. The average number of the generated virtual networks for each simulation ranges from 2000 to 4000, which are extremely large, making our results highly reliable. Therefore, there is no need to show confidence intervals.

2) Performance Metrics

We use the following most common metrics for performance evaluation.

- Acceptance ratio: This metric measures the percentage of the number of VN requests accepted during the given simulation time period.
- Average node utilization: This metric is computed by averaging the time-averaged utilizations over all substrate nodes.
- Average link utilization: This metric is the average of the time-averaged utilizations of all substrate links.
- Average number of colocated nodes: This metric is calculated by averaging the time-averaged number of colocated nodes over all SSNs/STNs.
- Average number of mapped nodes: This metric is calculated by averaging the number of mapped nodes over all VSNs/VTNs.

3) Comparative Results

We consider two major factors that may affect the performance of VN embedding. One is the VN arrival rate that decides the load of the substrate network. The other is the

mapping radius between the locations of virtual nodes and their candidate nodes. We set the mapping radius to be the same for all virtual nodes of a VN to simplify parameter setting.

We first look at the results with varying VN arrival rates from average 4 to 8 arrivals per 100 time units while the mapping radius D of VNs is set to 10. Fig. 8(a) shows the acceptance ratio vs. VN arrival rate. While both D-ViNE and our ACNV show similar trend with increasing load, the acceptance ratios of ACNV are always higher than the ones of D-ViNE. From Fig. 8(b), we can see that the average node utilization of ACNV is higher than D-ViNE. This is not a surprise. Although all three mapping options (one-to-one, splitting, colocation) do not reduce actual CPU resource usage for a single virtual node, with higher acceptance ratio, the total accepted CPU load is higher for ACNV. Therefore the average node utilization is higher, which clearly indicates the reduction of fragmented resources.

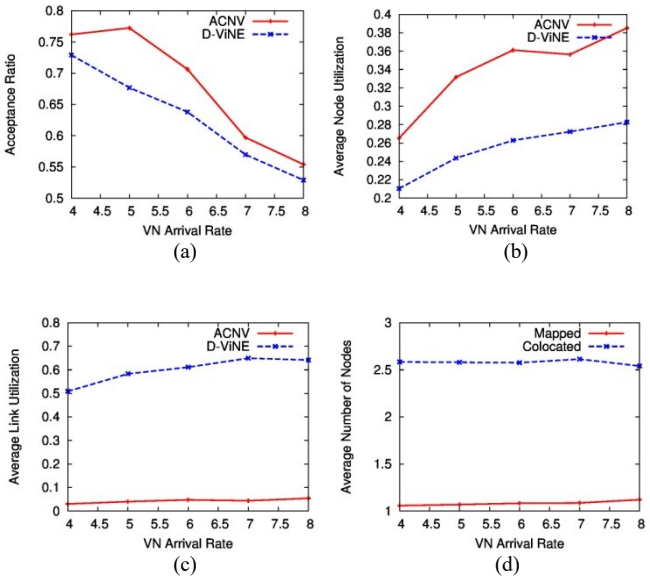

Fig. 8. (a) Comparison of acceptance ratios for ACNV and ViNE under varying VN arrival rate; (b) Comparison of average node utilizations of ACNV and ViNE with increasing VN arrival rate; (c) Comparison of average link utilizations of ACNV and ViNE with increasing VN arrival rate; (d) Average number of substrate nodes a virtual node is mapped onto and average number of colocated virtual nodes with increasing VN arrival rate.

On the other hand, average link utilization behaves completely different as shown in Fig. 8(c). Here the average link utilization of ACNV is significantly lower than D-ViNE. This clearly shows the gain of collocation by reducing communication overhead. Lower link utilization and higher acceptance ratio will certainly bring higher revenue for InPs.

We now study the inter-relationship between node splitting and colocation. The objective function as formulated in Eq. (7) depends on both node and link utilizations. Because CPU loads cannot be reduced through either node splitting or colocation for a single virtual node, the objective function is targeted at balancing loads in terms of CPU resources although the gain is not significant. Node splitting can certainly be helpful in achieving load balance. On the other hand, link utilizations can be reduced significantly through colocation. Therefore the objective function will strongly favor colocation. Nevertheless, because a virtual node can be both split and colocated, node splitting can provide more options for colocation through partial colocation and therefore be helpful in reducing bandwidth requirements. Fig. 8(d) shows the average number of substrate nodes a virtual node is mapped onto and the average number of colocated nodes. From Fig. 8(d) we can see that the average number of colocated nodes is around 2.5 to 2.6 while the average number of nodes a virtual node is mapped onto is around 1.1. Both are relative stable with increasing load. These results show that, with modest colocation and node splitting, significant link bandwidth saving can be achieved.

The opportunities that a virtual node can be mapped to multiple substrate nodes and multiple virtual nodes can be colocated partially or fully certainly depend on the mapping radius. The larger the mapping radius is, the higher the number of candidate nodes is. To test the impacts of mapping radius, we varied the mapping radius from 2 to 15 while fixing the average arrival rate to 8 arrivals per 100 time units. Fig. 9(a) shows how the average number of colocated nodes and average number of mapped nodes of our ACNV scheme change with increasing mapping radius. Clearly, the average number of colocated nodes benefits more than the average number of mapped nodes due to the drive to minimize link usage. Although the average numbers are relative small, we found the maximum numbers can be as large as 5 for the number of mapped nodes and 9 for the number of colocated nodes, both are quite large for the small substrate network we have setup.

Figs. 9(b) and 9(c) show the average node and link utilization, respectively. Fig. 9(d) demonstrates the acceptance ratio. From these three figures we can see that, when the mapping radius is small, there are very few candidate substrate nodes for each virtual node. In most cases, VN requests are simply rejected because one or more virtual nodes cannot find any candidate node at all. Node splitting and colocation will not be helpful under this kind of situations. The results of ACNV and D-ViNE are nearly the same.

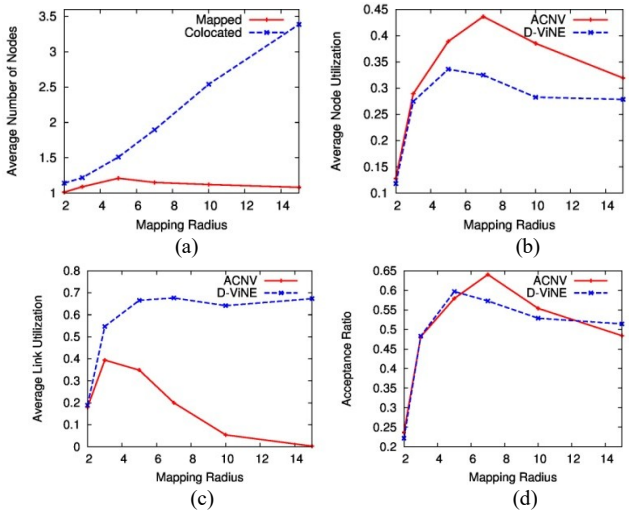

Fig. 9. Results with homogeneous nodes: (a) average number of mapped nodes and average number of colocated nodes; (b) comparison of average node utilizations of ACNV and ViNE; (c) comparison of average link utilizations of ACNV and ViNE; (d) comparison of acceptance ratios for ACNV and ViNE.

When the mapping radius is increased, acceptance ratios for both ACNV and D-ViNE increase dramatically because more virtual nodes can find candidate nodes. When acceptance ratios reach above 55%, lacking of link or node resources becomes the key factor instead of the number of candidate nodes. Here colocation can play a very important role by reducing the link bandwidth usage. It can be seen that ACNV shows consistent higher average node utilization because more virtual nodes are admitted and lower link utilization because of the benefits of colocation.

It is interesting to note that, when $D = 15$, the acceptance ratio of D-ViNE is actually higher than ACNV. This seems to contradict with the average node utilization in Fig. 9(b). As observed in [10], acceptance ratio as a metric can sometimes be misleading because an InP can accept a large number of VNs with low resource requirements and reject a small number of VNs with heavy resource requirements. Because D-ViNE only supports one-to-one mapping, a VN request with high CPU requirements will likely be rejected if any one virtual node of the VN request cannot find a candidate substrate node that has enough CPU resource to host the virtual node, while under ACNV, the virtual node can be split and accommodated through multiple substrate nodes. This situation typically happens when CPU resource is the main bottleneck part. Large mapping radius means more candidate nodes that lead to more alternative paths. For D-ViNE, bottleneck moves from link bandwidths to CPU resources with increasing number of alternative paths. This explains the differences between Figs. 9(b) and 9(d).

B. Heterogeneous Node Model

In this subsection, we consider heterogeneous node model where VSNs have CPU resource requirements while VTNs have transit bandwidth requirements. SSNs are limited by CPU capacities while STNs are not limited by either CPU capacities or internal transit bandwidths (i.e., STNs are internally non-blocking) as we discussed in Section II. Each substrate link has a bandwidth capacity limitation and each virtual link has a bandwidth requirement.

1) Simulation Setup

A substrate network was generated using GT-ITM tool with a transit-stub graph which includes one domain with 25 transit nodes in 25×25 grids. The links connecting transit nodes were generated in pure random with probability 0.3. Each transit node is attached with a stub node in 5×5 grids around its attached transit node. There were in total 50 nodes that included 25 transit nodes and 25 stub nodes and 121 links. Link bandwidth capacities were generated uniformly between 50 and 100. CPU capacities of stub nodes were also generated uniformly between 50 and 100.

VN requests were also created using GT-ITM tool with transit-stub graphs. Each graph had one domain with a random number of VTNs, which range from 2 to 10 nodes with an average of 5 nodes in 25×25 grids. The virtual link between each pair of virtual nodes was generated randomly with probability 0.8. Each VTN was attached with a VSN which was located in 5×5 grids around its attached VTN. The CPU requirement of each VSN was generated randomly between 0 and 15. VN arrival patterns are similar to the ones described in last subsection.

2) Transit Bandwidth Requirements

It is difficult to choose distribution ratios and bandwidths of virtual links while still satisfying Eq. (10) for the average 2000 to 4000 VNs we generated. Therefore we decided to setup the bandwidths of virtual links randomly between 0 and 3. Instead of defining different transit bandwidth requirements for different virtual links, we can define a uniform transit bandwidth requirement of a VTN based on a minimum (or maximum) bandwidth requirement for each virtual link incident to the VTN. Specifically, we define the following minimum (or maximum) virtual link bandwidth requirement for each VTN:

$$B^m(n^{VT}) = \min_{l^V \in L(n^{VT})} B(l^V) \tag{24}$$

Accordingly, we can define corresponding scaling factor for each virtual link as follows:

$$S_i^m(n^{VT}) = \frac{B^m(n^{VT})}{B(l_i^V)} \tag{25}$$

We will drop the parts in bracket wherever the context is clear.

Other than bandwidths of virtual links, distribution ratios also have impacts on transit bandwidth assignment as we discussed in Section II. In our simulation setup, we assumed a uniform traffic pattern which is widely adopted for typical switch design. Let $\gamma(n^{VT})$ denote the degree of node n^{VT}. Then we set

$$\alpha_{ij}(n^{VT}) = \frac{1}{\gamma(n^{VT}) - 1}, \forall i \neq j | l_i^V, l_j^V \in L(n^{VT}) \tag{26}$$

Eq. (26) implies a VTN will distribute the incoming traffic from a virtual link equally to all outgoing virtual links.

With Eqs. (24) and (26), from Eq. (7), we have

$$\beta_{ij}(n^{VT}) = \frac{1}{\gamma(n^{VT}) - 1}, \forall i \neq j | l_i^V, l_j^V \in L(n^{VT}) \tag{27}$$

With the above treatments, we can adapt Eq. (19) as the following:

$$\frac{S_j^m f_{\underline{k}u}^j}{\gamma(k) - 1} + \sum_{v \in N^C(k)} f_{uv}^{ijk} - \frac{S_i^m f_{u\underline{k}}^i}{\gamma(k) - 1} - \sum_{v \in N^C(k)} f_{vu}^{ijk} \geq 0,$$
$$\forall u \in N^C(k), \underline{k} = \mu(k), k \in N^{VT}, l_i^V, l_j^V \in L(k) \tag{28}$$

Our simulation with heterogeneous nodes were made based on the LP formulation in Section II with Eq. (19) replaced by Eq. (28). These modifications are used here simply for easy simulation parameter setup. If internal congestion is a concern, maximum value instead of minimum value can be used (may cause congestion at output ports). They do not affect the general applicability of Eq. (19).

3) Simulation Results

Similar to the simulation with homogeneous nodes discussed earlier, we considered the impacts of load and mapping radius.

We first check the results with mapping radius fixed to 10. Fig. 10(a) shows the average CPU utilization of substrate stub nodes and the average utilization of substrate links with increasing VN arrival rate. Fig. 10(b) shows the acceptance ratio. From these figures, we can see that both stub node CPU utilization and link utilization increase with increasing load while acceptance ratio decreases.

The gain of node splitting and colocation is not as visible as homogeneous node scenario discussed earlier. The reason behind this is the limitation imposed on different types of nodes. Because we only allowed virtual nodes to be mapped onto the same type of substrate nodes, i.e., virtual stub nodes to substrate stub nodes and virtual transit nodes to substrate transit nodes, the numbers of candidate nodes have reduced dramatically. So the opportunities of splitting and colocation have reduced. This can be confirmed in Figs. 10(c) and 10(d), where Fig. 10(c) shows the average numbers of mapped and colocated stub nodes and Fig. 10(d) shows the average numbers of mapped and colocated transit nodes. They are all below 2.

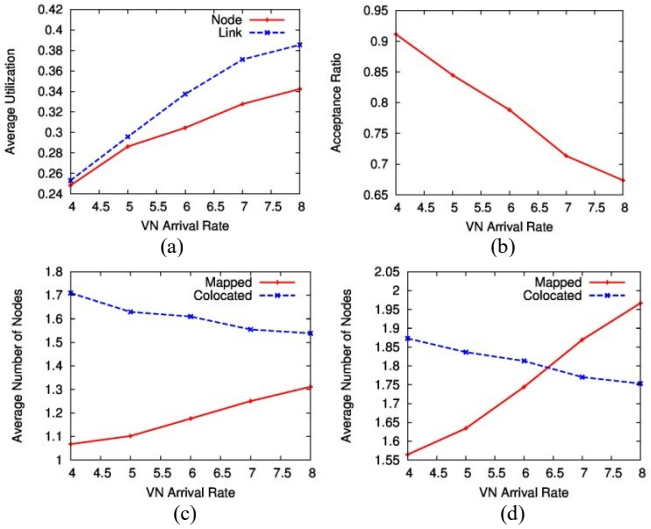

Fig. 10. (a) Average CPU utilization of substrate stub nodes and average link utilization of substrate links with increasing VN arrival rate; (b) Acceptance ratio with increasing VN arrival rate;(c) Average number of substrate stub nodes a virtual stub node mapped onto and average number of colocated virtual stub nodes with increasing VN arrival rate; (d) Average number of substrate transit nodes a virtual transit node mapped onto and average number of colocated virtual transit nodes with increasing VN arrival rate.

Furthermore, when load increases, the average numbers of mapped nodes increase while the average numbers of colocated nodes decrease. This result indicates saturation of both the CPU capacities of stub nodes and the transit bandwidth capacities of transit nodes as also confirmed by the dramatic drop in acceptance ratio in Fig. 10(b).

One way to increase the number of candidate nodes is to increase the mapping radius. We changed the mapping radius from 2 to 15 while fixing the average arrival rate to 8 arrivals per 100 time units. Figs. 11(a) and 11(b) show the average numbers of mapped and colocated stub nodes and the average numbers of mapped and colocated transit nodes with increasing mapping radius. While both node splitting and colocation increase with increasing mapping radius, colocation increases more dramatically with stub node. As shown in Fig. 11(c), the average link utilization starts decreasing when mapping radius increases beyond 10, which is in coincidence with dramatic colocation increase after mapping radius reaches beyond 10 as shown in Figs. 11(a) and 11(b). This clearly shows the benefits of colocation.

The average node utilization shows saturation after radius reaches 10. Acceptance ratio as shown in Fig. 11(d) slightly drops after mapping radius is beyond 10. This is again because the stub nodes accepted more VN requests with higher CPU demands and blocked a large number of VN requests with smaller CPU demands when stub nodes become saturated.

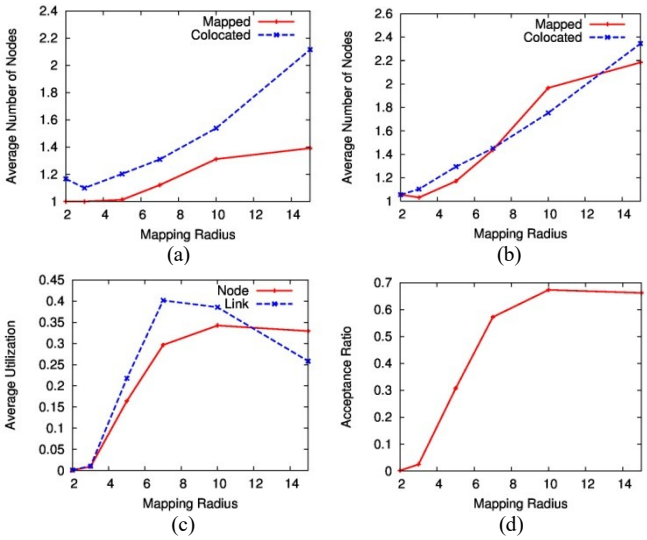

Fig. 11. Results with heterogeneous nodes: (a) average number of SSNs a VSN mapped onto and average number of colocated VSNs; (b) average number of STNs a VTN mapped onto and average number of colocated VTNs; (c) average CPU utilization of sub substrate nodes and average link utilization of substrate links; (d) acceptance ratio.

IV. Related Work

Network virtualization is a relatively new concept. [6] and [7] provide comprehensive surveys related to this area. In the following part, we focus on those works that are closely related to this paper.

An earlier study [8] considered the one-to-one online mapping scenario where VN requests arrive and leave dynamically. This kind of network embedding problem is NP-hard. Heuristic algorithms developed in the paper measures the potential of each substrate node based on the stress of the node and the stress of its associated links. Virtual nodes are then assigned to substrate nodes according to the potential of each substrate node. After nodes are assigned, links are assigned using shortest path algorithm.

Another research study [9] extends the paper [8] by allowing the substrate network to split a virtual link over multiple substrate paths. This helps reduce the link mapping part to MCF Problem with each virtual link as a commodity, which can be solved in polynomial time. Path splitting enables better resource utilization by avoiding bandwidth fragmentation. It also provides better load balance and reliability under dynamic network environments. However, the

node mapping part was still one-to-one which is NP-hard and heuristic algorithms were developed in [9].

A recent study [10] tried to solve the NP-hard node mapping problem by first relaxing the integer constraints for the node mapping to obtain a linear program and then applying rounding techniques to select unique node mapping. While this approach makes node mapping solvable in polynomial time, applied rounding techniques can also make the results suboptimal.

A latest work [11] on network virtualization extends [10] to WDM (Wavelength Division Multiploexing) and flexible-grid optical networks. The problem was formulated as mixed integer linear programs (MILP). Two heuristics were developed to achieve suboptimal solutions by exploring the opportunities of traffic grooming.

Another extension achieved in [21] is to implement survivable virtual network embedding by considering physical networks that may partially fail. Heuristic algorithms based on fast re-routing were developed to mitigate failures.

Inspired by the success of VMs enabled by Hypervisor, there are some recent works that started considering the colocation scenario [12]. A complex heuristic algorithm was developed to find optimal colocation strategy. However the work in [12] has not considered the case in which a node can be split and mapped onto multiple nodes. Nor has it considered differentiating transit nodes from stub nodes.

[2] and [3] introduced the VDC concept where a node can be either a VM or a virtual switch. While VM is considered non-splittable, splitting a virtual switch to multiple physical switches is investigated under two specific virtual topologies called virtual cluster and oversubscribed virtual cluster in [3]. It demonstrates the benefit of defining a virtual network with virtual switches for mitigating congestion.

[4] and [5] further studied the migration and reliability issues under the VDC model. However they limit the mapping from a virtual node to a physical node to one-to-one for both VM and virtual switch.

Our approach in this paper is different from all existing studies in the following aspects:

1. We defined a new abstraction for the interface between SPs and InPs. In our abstraction, an application is modeled as a virtual network of functional nodes rather than a virtual network of VMs. Our abstraction helps SPs to define their applications with models that match their software architectures closely. Meanwhile, it provides InPs more freedom to decide the number of VMs and their sizes for each application based on a global optimization process. This will help InP to auto scale each application based on load condition.
2. Our abstraction naturally leads to node splitting scenario that has not been considered by existing research studies on virtual network embedding. Our approach discussed in this paper allows many-to-many mapping that can significantly maximize the benefits of parallel computing.
3. We provided a generic way to map a VTN to multiple STNs. Specifically, we allow an SP to specify a traffic pattern for a VTN in addition to potential STNs as a requirement so that the optimization process can embed the requirement onto the STNs. This allows an InP to provision its network more accurately for different applications while giving SPs more control.
4. Our formulation leads to LP problems that can be solved in polynomial time. This is an advantage over existing approaches that require some kind of heuristics in order to achieve polynomial time, which lead to suboptimal results.

V. Conclusion

Existing research studies on application modeling have put limitations on how a virtual node can be mapped onto a substrate node. These limitations have reduced the benefits of resource sharing and increased the difficulty of resource optimization.

In this paper, we have developed a novel approach that allows node splitting as well as node colocation either partially or fully. Furthermore, we have considered a more realistic VN mapping scenario where both virtual nodes and substrate nodes are classified as either transit node or stub node. For VTNs, we introduced a new requirement called traffic pattern. These improvements allow us to model all applications supported by today's datacenters, especially new programming models such as MapReduce, which explore a large cluster of servers in parallel. They will also be helpful to mitigate congestion problems caused by poor provisioning.

We formulate virtual links and virtual transit links as two levels of commodities. Our formulation results in LP problems that are more scalable than all existing approaches. Through simulation, we have demonstrated our approach can dramatically reduce link bandwidth consumption and raise VN acceptance ratio. Our approach leads to higher node utilization that indicates reduction of resource fragmentation.

Future research will be targeted at extending our model to capture migration and energy cost.

References

[1] J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," *USENIX Sixth Symposium on Operating System Design and Implementation* (*OSDI'04*), San Francisco, CA, December, 2004

[2] C. Guo, G. Lu, H. J. Wang, S. Yang, C. Kong, P. Sun, W. Wu, and Y. Zhang, "Secondnet: a data center network virtualization architecture with bandwidth guarantees," In *Proceedings of the International Conference Co-NEXT,* 2010.

[3] H. Ballani, P. Costa, T. Karagiannis, and A. Rowstron, " Towards predictable datacenter networks," In *Proceedings of the ACM SIGCOMM 2011*, New York, NY, USA, 242-253.

[4] M. F. Zhani, Q. Zhang, G. Simon, and R. Boutaba, "VDC Planner: Dynamic Migration-Aware Virtual Data Center Embedding for Clouds," In *Proceedings of the IFIP/IEEE Integrated Network Management Symposium (IM 2013)*, Ghent, Belgium, May 2013.

[5] Q. Zhang, M. F. Zhani, M. Jabri, and R. Boutaba, "Venice: Reliable Virtual Data Center Embedding in Clouds," In *IEEE International Conference on Computer Communications (INFOCOM)*, Toronto, Ontario, Canada, April 27 - May 2, 2014.

[6] M.F. Bari, R. Boutaba, R. Esteves, L.Z. Granville, M. Podlesny, M.G. Rabbani, Q. Zhang, M.F. Zhani, "Data Center Network Virtualization: A Survey," *IEEE Communications Surveys and Tutorials*, vol.15, no. 2, Sept. 2012.

[7] A. Wang, M. Iyer, R. Dutta, G. Rouskas, I. Baldine, "Network virtualization: technologies, perspectives, and frontiers," *IEEE Journal of Lightwave Technology*, vol.31, no.4, pp.523–537, 2013

[8] Y. Zhu and M. Ammar "Algorithms for assigning substrate network resources to virtual network components," *IEEE INFOCOM*, April 2006, Barcelona.

[9] M. Yu, Y. Yi, J. Rexford, and M. Chiang, "Rethinking virtual network embedding: susbtrate support for path splitting and migration," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 17-29, April, 2008

[10] N. M. M. K. Chowdhury, M. R. Rahman, and R. Boutaba, "Virtual network embedding with coordinated node and link mapping," *IEEE INFOCOM 2009*, April 2009, Rio de Janeiro.

[11] S. Zhang, L. Shi, C. S. K. Vadrevu, B. Mukherjee, "Network virtualization over WDM and flexible-grid optical networks," *Optical Switching and Networking*, Elsevier, vol. 10, no. 4, pp. 291-300, Nov. 2013.

[12] C. Fuerst, S. Schmid, and A. Feldmann, "Virtual Network Embedding with Collocation: Benefits and Limitations of Pre-Clustering," *IEEE International Conference on Cloud Networking*, November 11-13, 2013, San Francisco.

[13] http://aws.amazon.com.

[14] J. Varia, "Best Practices in Architecting Cloud Applications in the AWS Cloud," *Cloud Computing: Principles and Paradigms*, Wiley, 2011.

[15] D. Sullivan, "PaaS Provider Comparison Guide: Amazon AWS as a PaaS," http://www.tomsitpro.com/articles/amazon-aws-paas-iaas-cloud-computing,2-608.html, September, 2013.

[16] M. Alizadeh, A. Greenberg, D.A. Maltz, J. Padhye, and et al., "Data Center TCP (DCTCP)," in *Proceedings of SIGCOMM'10*, August, 2010, New Delhi, India.

[17] D. Nagle, D. Serenyi, and A. Matthews, "The Panasas ActiveScale Stirage Cluster: Delivering Scalable High Bandwidth Storage," *2004 ACM/IEEE Conference on Supercomputing*, 2004, DC, USA.

[18] A. Phanishayee, E. Krevat, V. Vasudevan, D. G. Andersen, G. R. Ganger, G. A. Gibson, and S. Seshan, "Measurement and Analysis of TCP Throughput Collapse in Cluster-Based Storage Systems," *USENIX Conference on File and Storage Technologies*, February 2008, San Jose, CA.

[19] "GNU Linear Programming Kit," http://www.gnu.org/software/glpk/.

[20] E. Zegura, K. Calvert, and S. Bhattacharjee, "How to model an Internetwork," in *Proceedings of IEEE INFOCOM*, 1996, pp. 594–602.

[21] M. R. Rahman and R. Boutaba, "SVNE: Survivable Virtual network Embedding Algorithms for Network Virtualization," *IEEE Transactions on Network And Service Management*, vol.10, No.2, June 2013, pp.105-118.

Changcheng Huang received his B. Eng. in 1985 and M. Eng. in 1988 both in Electronic Engineering from Tsinghua University, Beijing, China. He received a Ph.D. degree in Electrical Engineering from Carleton University, Ottawa, Canada in 1997. From 1996 to 1998, he worked for Nortel Networks, Ottawa, Canada where he was a systems engineering specialist. He was a systems engineer and network architect in the Optical Networking Group of Tellabs, Illinois, USA during the period of 1998 to 2000. Since July 2000, he has been with the Department of Systems and Computer Engineering at Carleton University, Ottawa, Canada where he is currently a full professor. Dr. Huang won the CFI new opportunity award for building an optical network laboratory in 2001. He is an associate editor of *Springer Photonic Network Communications*. Dr. Huang is a senior member of IEEE.

Jiafeng Zhu is a Sr. Architect in Shannon Lab at Huawei Technologies, Inc. U.S. R&D Center. His research interests include distributed processing scheduling, cloud storage, software define networking, network function virtualization, and network planning.