Joint Virtual Middlebox and Application Placement with Bandwidth Guarantees in Multi-tenant Datacenter Networks

***Abstract*—Hardware middleboxes are widely used in current cloud datacenter to provide network functions such as firewalls, intrusion detection system, load balancers, etc. Unfortunately, they are expensive and unable to offer customized functions for individual tenant. To overcome this issue, there is an increasing interest in deploying software middleboxes to enable flexible security, network access functionality. A key problem for cloud providers is how to efficiently deploy the software middleboxes and provide performance guarantee for both middleboxes and applications while saving as many resources as possible. Besides, it is necessary to conduct an effective scaling approach to accommodating the middlebox scaling request when the allocated middleboxes cannot handle the practical workload well. In this paper, we present *vMAP*, a solution that combines a tenant request model with an effective placement algorithm for middlebox and application VMs, while conducting a probability model based approach to addressing the middlebox scaling problem. We first specify the model of the tenant's requirement that matches the demands for virtual machines of the middlebox and application, as well as communication traffic. A joint virtual middlebox and application placement algorithm is then proposed to offer predictable network performance for each tenant, and minimize datacenter bandwidth utilization. Finally, we provide an approach to solving the middlebox scaling problem through reserving additional free virtual machine (VM) resource based on a probability model. Extensive simulation results based on current large-scale datacenter networks verify that vMAP can accept more tenant requests than existing placement algorithm (e.g., Stratos [1]) while saving ∼10% core-level bandwidth. The middlebox scaling experiments also show that our middlebox scaling approach can accommodate the scaling request effectively while minimizing the waste of the reserved virtual machine resource.**

I. Introduction

Large public clouds, such as Amazon EC2 and Windows Azure, provide infrastructure as a service (IaaS) for enterprises and individuals. Besides the ability to provide computing resources, public clouds are expected to support network functionality, such as load balancers (LB), firewall (FW), intrusion detection system (IDS), etc. Currently, network functions are deployed through the expensive hardware middleboxes (in short, MBs) which are placed at the point close to the data center switches. Since hardware middleboxes are shared by all tenants, it is hard to manage these expensive devices to provide customized functions for each tenant. Besides, heavy congestion may happen at links around those middleboxes. To address these issues, there is an increasing trend towards deploying software middleboxes (or, interchangeably, virtual middleboxes, middlebox VMs) in the cloud [2], [1].

Providing virtual middleboxes with guaranteed performance is challenging for cloud service providers due to several reasons. First, we need to model various requirements of tenants, which consist of application VMs, middlebox VMs as well as the connecting network. Second, it requires an efficient algorithm to place virtual middleboxes in cloud service infrastructure to maxmize the datacenter resource utilization by accepting as many requests as possible. Last but not least, the performance of software MBs is not easy to predict in real environment due to many factors like the number of rules, algorithms and implementations. The experience based selection of MBs may under- or over-estimate resource consumption.

In this paper, we propose *vMAP* (short for *v*irtual *M*iddlebox and *A*pplication *P*lacement), a solution that combines a tenant request model with an effective placement algorithm for both middlebox and application, while accommodating the middlebox scaling request through a probability model based approach. Tenant's requirement is represented by a weighted directed acyclic graph (DAG). The nodes consist of diverse virtual middleboxes and a set of homogeneous application VMs, both of which are dedicated for each tenant to process arrival traffic. The edges of DAG indicate the communication requests of middlebox-to-middlebox, middlebox-to-application and intra-application. The weight on the edge is the minimum communication requirement between VMs of the middlebox and application [1], [3]. The inter-tenant communication [4] is also structured by a directed graph. The arrival traffic from other tenants will be forwarded to the MBs of the target tenant before arriving at its application VMs.

Existing placement algorithm like Stratos [1] seeks to partition the VMs into per-rack partitions and explores to assign partitions to racks with minimal inter-partition communication. However, the location-unawareness rack selection in Stratos leads to bandwidth waste of core-level links, which are the bottlenecks for most oversubscription data center networks. More Specifically, Stratos adopts a greedy approach to sorting pairs of partitions in the decreasing order of the inter-partition communication and finds a pair of racks with the highest available bandwidth to accommodate these two VM partitions. The inter-partition communication may travel through several switches even the communicating partitions can be placed under the same switch, since Stratos only considers the bandwidth resource apart from the locations of the racks selected to accommodate the partitions. In contrast, to save the core-level bandwidth, we first traverse the datacenter topology in

a bottom-up manner to search for a valid lowest level sub-tree [3] that has enough VM slots for the tenant request. Then we exploit to carefully place the required MBs and application under the sub-tree with the minimal bandwidth consumption. The key idea is to eliminate unnecessary communication across racks or sub-trees due to the mismatched capacities between the upstream and downstream middlebox or application VMs. To this end, we deploy middlebox and application VMs in proportion to reduce the traffic on the link. Further, we try to co-locate more application VMs to reduce the intra-application communication cost through carefully exchanging the positions of middlebox and application VMs.

It is notable that the performance of middlebox is hard to predict in practice. The experience based middlebox selection may cause under-/over-estimation of the resource consumption in the shared environment and affect the performance of deployed applications. Hence, it is necessary to conduct a middlebox scaling approach when the capacity of allocated middleboxes cannot handle the practical workload well. To address the scalability requests of tenants, we adopt an approach of pre-allocating a few additional free VM slots. The pre-allocating approach is helpful for fast and localized deployment of scaled middlebox instances, which can improve the application performance and reduce the bandwidth waste.

To the best of our knowledge, we are the first to model the virtual middlebox network with intra- and inter-tenant communication. Extensive simulations on large-scale datacenter topology have shown the effectiveness of our VM placement algorithm. Compared with the existing placement algorithm, our algorithm can greatly reduce the bandwidth consumption of core-level links while making the best use of low-level links' bandwidth resource. The preserved middlebox scaling approach can satisfy the scaling requirement effectively while reducing the cost of re-allocating the middlebox chain for the accepted tenant. Generally, the network function virtualization enables the cloud service provider to offer customized services with lower cost and our study will be helpful to build more flexible software-based cloud services to meet diverse requirements of the tenants in the datacenter network.

The rest of this paper is organized as follows. Section II reviews the background, related works and challenges of deploying tenant requests effectively in datacenter networks. Section III describes our abstraction of middlebox and the communication model. The VM placement strategy is explained in Section IV. The scalability problem is discussed in Section V, and the simulation based performance evaluation is demonstrated in Section VI. Section VII concludes this paper.

II. Background and Challenges

A. Background and Related Works

Multi-tenant datacenter. Large-scale datacenter is the core infrastructure for cloud based services. Datacenter network is organized in a multi-level tree-like topology. See Fig. 1, each rack typically contains 40+ servers, each of which is configured with a 10Gbps network interface card (NIC) connecting to the top-of-the-rack (ToR) switch. The ToR

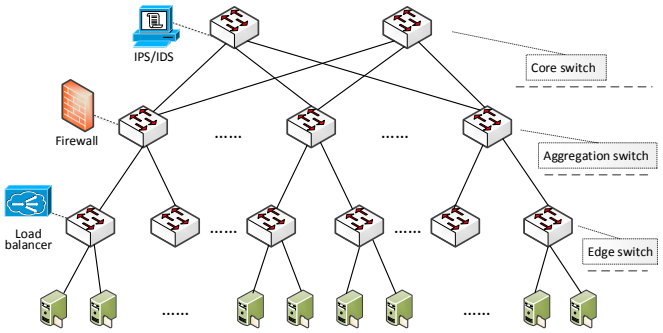


Fig. 1. Datacenter network and hardware middlebox deployment

switch will balance the traffic to multiple aggregation switches. All aggregation switches are further connected to the core switches. Note that the communication bandwidth between adjacent layers is oversubscribed by a ratio from 2:1 to 10:1, which may vary slightly with different configurations [4], [5], [6].

In multi-tenant datacenter environments, a tenant can rent multiple virtual machines (VMs) on physical servers. The VMs are configured with different amounts of CPU, memory and storage resources. From the network perspective, we usually model the computation and storage resources to be the number of slots requested in physical machines. All VMs assigned to a tenant are interconnected to form a virtual network (VN) such that VMs can communicate with each other to exchange data. The traffic inside a multi-tenant datacenter carries external traffic to and from the Internet, intra-application traffic between a tenant's application VMs and inter-tenant traffic. Specifically, the inter-tenant traffic consists of flows between tenants and flows between the tenants and the cloud service provider [4], which can also be considered as a special tenant. Inter-tenant communication increases with the increase of services offered by cloud providers. Based on recent measurement in public cloud provider, 65% of traffic happens between VMs belonging to one tenant VN and 35% is among different VNs or tenant-provider communication [4], [7].

Middleboxes in multi-tenant datacenter. Network functions are also necessary and provided in the form of hardware middleboxes in current datacenters. However, hardware middelboxes are known to have several drawbacks. For example, working as a shared resource, the hardware middleboxes can only provide common services used by all tenants instead of providing customized services for different tenants. Once the middlebox failed, all the tenants will be affected. Besides, the fixed locations of hardware middleboxes result in aggregated network bandwidth consumption and cause network congestion [8], [9].

An alternative solution to hardware middleboxes is network function virtualization (NFV) [10]. Deploying software middleboxes will greatly improve the flexibility, manageability and cost-efficiency. Software middlebox is not necessary to be shared and able to provide customized services with perfor-

mance guarantee for each dedicated tenant. It has been shown that the software middleboxes are able to offer equivalent functionalities as the corresponding hardware implementations [11], [12], [2], [8]. The advance of software defined networking (SDN) further facilitates the deployment and management of the software middleboxes running in datacenters [13], [14], [15], [16]. Although there remains unsolved challenges before it can be integrated into the cloud ecosystem, the perspective of deploying flexible software middleboxes for isolated tenants has motivated us to extend the previous works to realize predictable middlebox virtual networks through effective virtual middlebox placement.

Placement of application VMs with performance guarantee. Since the underlying network is shared by tenants, the application can be affected by variable and unpredictable network performance. A series of works have been proposed to enable traffic isolation and provide bandwidth guarantee, including Oktopus [3], FairCloud [17], ElasticSwitch [18], etc. The above works focus only on intra-tenant communication. The inter-tenant traffic isolation is addressed by Hadrian [4]. Through hierarchical hose model, the communication dependencies can be explicitly declared. The cloud provider then allocates resource according to the specification of tenants.

Placement of middlebox VMs with performance guarantee. Most of previous works considered the bandwidth guarantee for applications. As far as we know, few works address the bandwidth guarantee for virtual MBs in multi-tenant datacenters [19]. The NFV cloud computing platforms, like NFV OPEN LAB [20] and CloudNFV [21], provide an architecture for NFV deployment and management. Theoretically, to obtain effective use of resource, various NFV placement objectives have been studied, for example, remaining data rate, latency, number of used network nodes as well as distance and setup cost, see [22], [23] and the references therein. However, they could not provide bandwidth guarantee for tenants. Stratos [1] considered the middlebox placement in multi-tenant datacenters through a systematic way by avoiding network congestion. In contrast, we are the first to model the virtual middlebox network with intra- and inter-tenant communication and provide bandwidth guarantees for both middlebox and application.

B. Design Challenges

Deploying both virtual middlebox and application with performance guarantee is challenging for cloud service providers in datacenter networks due to several reasons following.

Model to specify the tenant's requirement. In order to effectively deploy the tenant requests, we need to accurately model the tenant requirements, which consists of middlebox, application, as well as the intra- or inter-tenant communication patterns. Besides, an intuitive and descriptive abstract model benefits both tenants and providers, which is not trivial to build. To address above problems, we adopt the *Tenant Application Graph (TAG)* [24] model and extend the Hadrian [4] model to properly match the tenant requirements. We introduce the modelling details in Section III.

Strategy to place both the middlebox and application. Given tenant requests and corresponding communication models, our objective is to maximize the datacenter resource utilization by accepting as many requests as possible. This is an NP-hard problem like optimally deploying the *hose model* [25], [24]. Therefore, we search for effective heuristic solution to maximize the datacenter resource utilization. We present our placement algorithm in Section IV.

Approach to accommodating the middlebox scaling request. It is a remarkable fact that the behavior of current software middleboxes is actually not easy to predict in the real system. The performance depends on multiple factors, including workload like traffic demands, type and number of rules, and middlebox design, e.g., algorithm design, systematical implementation. Therefore, the scalability of MBs is important and need to be considered by the cloud service provider. Once the virtual network is placed in datacenter, it is hard to find a proper resource to migrate MBs and re-allocate live flows across MBs with performance guarantee [2]. We discuss the scalability issue in Section V.

III. Tenant Network Abstraction

In a multi-tenant datacenter, by leveraging the network function virtualization (NFV), cloud providers provide middlebox as a fundamental network service to tenants, like both computing and storage resources. A tenant could specify their middlebox and application (i.e., computation/storage) requirements in forms of VMs and the expected minimum bandwidth requirement for intra- and inter-tenant communication. Cloud providers will then assign VMs for middleboxes (in short, MB VMs) and applications (in short, APP VMs) according to the tenant's request. The cloud providers also enforce the middlebox policies, i.e., the sequence of middleboxes that a flow must traverse. In this section, we explain the model for APP VMs and MB VMs. We then introduce the abstraction of virtual networks associated to one tenant.

A. Model of APP VMs

To provide predictable network performance for the communication of N APP VMs in one tenant, users specify the minimum bisection bandwidth B^{in} for individual APP VM. Analogous to previous models [4], [17], the requirement of the minimum bandwidth guarantee for each APP VM belonging to the same tenant can be abstracted by the *hose model* [25]. In the hose model (see the thick black arrows in Fig. 2, tenant P), the application VMs are connected to a logical switch by a link of capability B^{in}.

Similarly, for the cross tenant communication, a tenant can specify the bandwidth request for a single APP VM. The minimum bandwidth guarantee for inter-tenant traffic of each APP VM is defined by B^{ex}. We extend the VN model of Hadrian [4] to support multiple external links. If the current tenant communicates to multiple tenants. The minimum bandwidth for each communication pair is guaranteed to be B^{ex}. The communication pattern between tenants can be expressed by a directed graph. We use the definition of *communication*

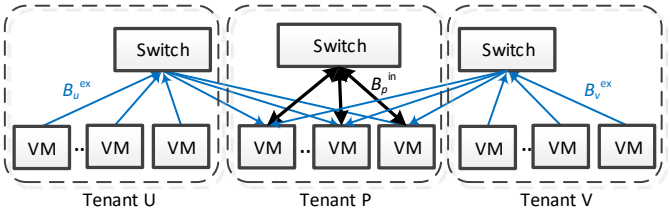


Fig. 2. Intra-tenant communication and inter-tenant communication

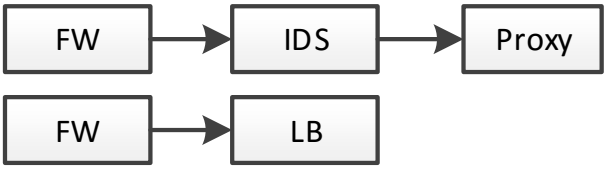


Fig. 3. Abstraction of MBs: Two MBs chains

dependency in [4] to explicitly specify the peer-to-peer traffic pattern. A tenant with a dependency of {*} means that he could allow any other tenants' communication request. We call this kind of tenant *open tenant* while the tenant who has communication requests to the *open tenant* is called *client tenant*. When the dependency is set as P:{U, V}, it means that the communications from tenant U and tenant V to tenant P are allowed. An example is illustrated in Fig. 2, the thin blue arrows show the communication between tenant U and tenant P, as well as tenant V and tenant P.

B. Model of MB VMs

Middleboxes perform different network functions to meet traffic demands of applications. A tenant may vary network functions simultaneously to process packets of different traffic classes, as shown in Fig. 3. Each request consists of a sequence of various MBs, such as firewall (FW), intrusion detection system (IDS), redundancy elimination (RE), load balancer (LB) and NAT Proxy, etc., which provide functionalities of access control and traffic engineering.

Generally, the communication between MBs can be abstracted by a directed-acyclic graph (DAG), where the vertex represents the type of middleboxes and the arc indicates the traffic flow between them. The external traffic should pass through the specified middleboxes one by one according to the DAG. The sequence can not be violated. Otherwise, the expected functionality will be ruined. In addition, we assume that the traffic still keeps the same when multiple MBs are introduced. This is reasonable because the traffic is the minimum bandwidth guaranteed for both intra- and inter-tenant communication. The normal traffic could pass all the rule checking and will not be blocked by middbleboxes like FW or IDS. For those middelboxes, e.g., RE or LB, which change the traffic load, they are usually placed at the end of

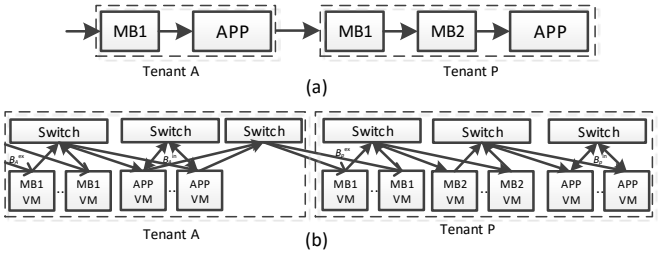


Fig. 4. Virtual Network Model. (a) the communication between two tenants; (b) cascade of TAG

the DAG. The variation of traffic is specified by the proposed parameters of B^{in} and B^{ex} in the aforementioned APP VMs model.

Therefore, a middlebox chain model can be defined by a tuple $\langle T, M\rangle$, where the type of MBs is denoted by a vector T, and the number of required MB VMs is denoted by a vector M. The direction of traffic flow is implicitly expressed by the sequence of items in the vector T. For instance, the first MB chain in Fig. 3 can be expressed as $\langle T, M\rangle$, where $T = \{\text{FW, IDS, Proxy}\}, M = \{2, 4, 2\}$ (we will present the computing process in detail to obtain a more accurate value of M in Sec. IV). It means that the tenant requests for 2, 4 and 2 VMs of FW, IDS, and Proxy respectively. Specifically, the sequence of middleboxes shows that the traffic will arrive at FW first, and then go to IDS and pass Proxy before reaching the application VMs. Besides, the tenant request can be classified into multiple MB chains associated with different applications [1], [22], [19] since a tenant may need to apply different middlebox policies to different traffic classes. See Fig. 3 as an example. The Internet traffic will pass Firewall, Intrusion Detection System and then NAT proxy, while the traffic from other tenants may only pass Firewall and Load Balancer. In this situation, the tenant's request can be specified by multiple middlebox chains' tuples of $\langle T, M\rangle$.

C. Virtual Network Model for MB VMs and APP VMs

We combine the MB model and APP model together to form the virtual network (VN) model to represent the tenant's request. The interaction between VMs of the same tenant is expressed by the *hose model* [25]. The middlebox chain should be attached to the ingress path of the external link of the VN. Such that, all external traffic coming into APP VMs must traverse through these specified middlebox chains which provide access control and other network functions. CloudMirror [24] provided a new network abstraction that allows the applications to specify their directional and specific communication patterns. We adopt a cascade of *Tenant Application Graph (TAG)* proposed in [24] to specify the VN model.

Fig. 4(a) shows an example, where a sequence of MBs is attached at the external link of APP VMs of tenants. The arrow shows the traffic flow. Arrival traffic of tenant A will pass

through all these middleboxes before being distributed to the application services. Similarly, the departure traffic of tenant A is forwarded to its destination (tenant P) which needs to pass through the middleboxes of tenant P first [1]. Fig. 4(b) uses the cascade of TAG model to abstract the communication dependency of VNs. The external traffic passing through each type of middleboxes in sequence are expressed by the dedicated directional arrows. The internal traffic of APP VMs is expressed by bi-directional arrows, each of which has a capacity of B_A^{in} (tenant A) or B_P^{in} (tenant P). The external traffic of each VM of tenant A (resp. tenant P) is bounded by B_A^{ex} (resp. B_P^{ex}).

In general, two kinds of traffic may go through the MBs before coming to the application VMs: 1) the Internet traffic; 2) the traffic from other tenants. When declaring a connectivity relationship, this external link can only communicate with those specified virtual networks (VNs). The unauthorized traffic will be blocked by MBs. Based on the models of APP VMs and MB VMs, a tenant can specify the connectivity of an external link which receives (or sends) traffic from any VN. In summary, a tenant's request can be specified by multiple virtual networks (or VNs) and each VN can be specified by a tuple $\langle N, B^{in}, B^{ex}, dependencies, T, M\rangle$.

IV. Virtual Network (VN) Deployment

Based on the tenant's request, the cloud provider allocates enough resources for MBs and APPs. We present a VM placement algorithm that bridges the gap between the high-level virtual network model of the middlebox and application to the low-level physical network. The VM request will be satisfied by providing enough free slots on physical machines (PMs). Since the network function demand consists of not only the bandwidth request but also the capability to process a specific class of traffic in order, we leverage the technique of software defined networking (SDN) by using a logically centralized controller to manage datacenter through a global view of the network topology. SDN based traffic engineering manipulates the underlying network routers to explicitly direct the traffic through each middlebox sequentially.

In this section, we first show the main idea of our placement algorithm. Next we analyze the bandwidth requirement (or bandwidth composition) for the tenant request. And then, we design a bandwidth saving placement strategy based on the bandwidth consumption analysis. Finally, we present our placement algorithm to efficiently deploy the VN model.

A. Overview of the Placement Strategy

For the placement of the tenant request, we aim to minimize the bandwidth consumption and save as much network resource as possible for newly arriving tenants. Instead of solving the placement problem as a graph partitioning problem [1], we seek to analyze the composition of bandwidth consumption in small granularity (inter-/intra- MB/APP VMs) and then plan to place the middlebox and application carefully to reduce the

[1]Some corporations may have the requirement to detect the departure traffic through additional middleboxes. We leave this for future work.

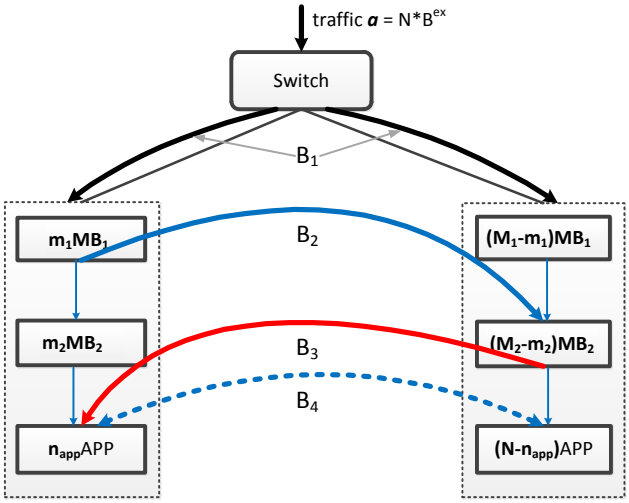


Fig. 5. Bandwidth requirement of $\langle N, B^{in}, B^{ex}, T, M\rangle$, where $T = \{MB_1, MB_2\}$, $M = \{M_1, M_2\}$

amount of bandwidth composition pertinently. Further, we find that: 1) placing different types of middleboxes with application VMs together or closely is an effective approach to reducing the bandwidth consumption of middlebox-to-middlebox and middlebox-to-application; And 2) co-locating application VMs is much beneficial for saving bandwidth of intra-application VMs. However, it is hard to leverage both placement principles simultaneously when deploying the tenant request due to the limited VM slot resource. Therefore, we seek to firstly consider enforcing one placement principle and then try to improve the other principle's gain by carefully exchanging the VMs in the leaf node of the deployed sub-tree.

B. Bandwidth Required by VN

Given a tenant request, the cloud provider needs to allocate sufficient bandwidth on the physical links to meet the bandwidth requirements specified in the VN model. We then analyze the bandwidth consumption to achieve a better understanding of the MB placement problem.

Consider a tenant request with a simple middlebox chain $\{MB_1, MB_2\}$ and assume that we have found two candidate sub-trees that have enough slot resource to hold the request as shown in Fig. 5. The total number of instances required by MB_k is indicated by M_k[2]. And m_k denotes the number of instances placed for MB_k in the left sub-tree. Correspondingly, the number of instances for MB_k in the right sub-tree is $M_k - m_k$. The numbers of APP VMs in the left and right sub-trees are denoted by n_{app} and $N - n_{app}$ respectively.

As shown in Fig. 5, the incoming traffic will pass the MB_1 and MB_2 successively and arrive at the tenant application

[2]We assume each instance of MB_k has a maximal traffic processing capacity c_k, so the minimum required number of MB_k is defined as $M_k = \left\lceil \frac{N \cdot B^{ex}}{c_k} \right\rceil$, where $N \cdot B^{ex}$ is the total incoming traffic.

finally. The traffic flows across and inside the sub-trees are illustrated in thick and fine arrows respectively. The middlebox splits the traffic proportionally to its downstream middlebox or application inside the same sub-tree (see the *thin arrows*). In the case that the traffic cannot be fully handled by the downstream middlebox or application inside the same sub-tree, it will be partly sent to the downstream middlebox or application outside the sub-tree (see the *thick arrows*). The bandwidth consumption across the sub-trees consists of four parts indicated by B_1, B_2, B_3 and B_4. The B_1 denotes the external traffic (represented by the black arrows) from the switch to MB_1 which is equal to $N \cdot B^{ex}$. The traffic that cannot be fully processed by MB_2 in the left will be partly sent to the instances of MB_2 in the right sub-tree as shown in the blue arrow labeled by B_2. Similarly, the traffic from MB_2 in the right to APP will be partly sent to the APP VMs in the left sub-tree (red arrow labeled by B_3) if the capacity of APP VMs in the right cannot serve the whole traffic from MB_2 in the same side. Besides, the bandwidth requirement (dotted arrow labeled by B_4) for intra-application communication is $2\min(n_{app}, N-n_{app})\cdot B^{in}$. For simplicity, we only consider the downlink bandwidth consumption since the uplink bandwidth has the similar characteristics. Then, by summing up the four parts of the bandwidth consumption, we can get the total downlink bandwidth requirement across the sub-trees:

$$\begin{aligned} B_{total} =& B_1 + B_2 + B_3 + B_4 \\ =& N \cdot B^{ex} + (m_1 \cdot b_1 - m_2 \cdot b_2) \\ &+ ((M_2 - m_2) \cdot b_2 - (N - n_{app}) \cdot B^{ex}) \\ &+ 2\min(n_{app}, N - n_{app}) \cdot B^{in}, \end{aligned} \tag{1}$$

where b_k is the required bandwidth[3] for each instance of MB_k.

For the general situation (i.e., the MBs and application VMs are randomly placed in the two sub-trees), by adding all the inter-middlebox traffic going across the sub-trees, we have the following equation:

$$\begin{aligned} B_{total} =& B_{switch,mb_1} + B_{mb_k,mb_{k+1}} + B_{mb,app} + B_{app} \\ =& N \cdot B^{ex} + \sum_{k=1}^{n-1} |m_k \cdot b_k - m_{k+1} \cdot b_{k+1}| \\ &+ |m_n \cdot b_n - n_{app} \cdot B^{ex}| \\ &+ 2\min(n_{app}, N - n_{app}) \cdot B^{in}. \end{aligned} \tag{2}$$

B_{switch,mb_1} is the incoming traffic for MB_1 from the switch which is equal to the external traffic demand. $B_{mb_k,mb_{k+1}}$ and $B_{mb,app}$ denote the incoming traffic for MB_{k+1} and APP VMs across the sub-trees from MB_k respectively. B_{app} represents the intra-application communication bandwidth requirement.

[3]For each VN model, the total incoming traffic is equally assigned to each instance of MB_k. And the required bandwidth b_k for each instance of MB_k can be calculated by $b_k = \frac{N \cdot B^{ex}}{M_k}$.

C. Bandwidth Saving Analysis

Finding beneficial placement principles. We try to save the total bandwidth requirement B_{total} by reducing the bandwidth consumption of the original four compositions of B_{total}. As illustrated above, the bandwidth composition $B_{switch\_mb_1}$ is a constant and cannot be saved. Thus, we consider to achieve bandwidth saving by reducing the bandwidth consumption of the remaining three compositions, i.e., $B_{mb_k,mb_{k+1}}$, $B_{mb,app}$, and B_{app}. More specifically, to reduce the traffic $B_{mb_k,mb_{k+1}}$ and $B_{mb,app}$ crossing the sub-trees caused by the mismatched capacities between the upstream and downstream MBs or APPs, we can place the MBs and APPs proportionally in the sub-tree according to their traffic processing capacity. In addition, with the increase of n_{app}, the intra-application bandwidth B_{app} increases from zero to $2 \cdot (N/2) \cdot B^{in}$ when $n_{app} = N/2$ and then decreases to zero. Thus, we can greatly save the intra-application bandwidth B_{app} through co-locating as many APP VMs as possible in one sub-tree. To sum up, we have two beneficial placement principles: 1) *placing different types of MBs with APP VMs together proportionally*[4] *is an effective approach to reducing the bandwidth consumption of* $B_{mb_k,mb_{k+1}}$ *and* $B_{mb,app}$, and 2) *co-locating APP VMs in one sub-tree is much beneficial to save the intra-application bandwidth requirement* B_{app}.

Enforcing the beneficial placement principles. However, the two beneficial placement principles cannot be always enforced simultaneously due to the limited available VM slot resource. As illustrated in Fig. 5, the VMs for middlebox and application of the tenant request have to be distributed among the two sub-trees if one sub-tree cannot accommodate the full request. Specifically, the APP VMs have to be placed in both two sub-trees when enforcing the first principle, which will conflict with the second principle. When co-locating all APP VMs in one sub-tree, the remaining available slots in the same sub-tree will be unable to hold the MB VMs and we have to place the MB VMs in another sub-tree, which violates the first principle but reduces the overall bandwidth consumption.

Combining the placement principles. Guided by the observations above, we seek to firstly consider enforcing the *principle 1* and then try to improve the benefit of *principle 2*. Specifically, to increase the degree of APP co-location in one sub-tree (enforce *principle 2*), we can exchange the positions of MB and APP in two sub-trees when *principle 1* is already enforced. The position exchanging may lead to mismatched capacities between the upstream and downstream MBs or APPs and produce extra traffic across sub-trees. The bandwidth requirement $B_{mb_k,mb_{k+1}}$ or $B_{mb,app}$ then has an increment. On the contrary, we have a reduction of intra-application bandwidth requirement B_{app} because of more APP

[4]Assume the total free VM slots in the left sub-tree is N_{slot}, then the number of MB_k needed to be placed in the left can be calculated by $m_k = \left\lfloor \frac{M_k}{\sum_{i=1}^{n} M_i + N} \cdot N_{slot} \right\rfloor$. The number of APPs needed to be placed in the left is computed as $n_{app} = \left\lfloor \frac{N}{\sum_{i=1}^{n} M_i + N} \cdot N_{slot} \right\rfloor$. Correspondingly, the numbers of MB_k and APP placed in the right are given as $M_k - m_k$ and $N - n_{app}$ respectively.

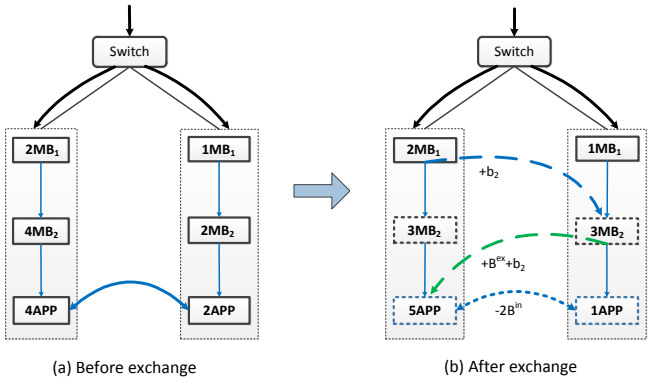


Fig. 6. Illustration of the bandwidth variation after exchange

VMs co-located. Therefore, we need to carefully select MB to exchange with APP to achieve the benefit of exchanging.

Selecting proper middleboxes to exchange. We then analyze the variation of the total bandwidth consumption before and after the VMs exchanging, which guides the selection of the proper MB VMs to exchange. The exchange may violate the proportion of the numbers of MB and APP VMs in the same subtree, which will cause extra flows from one side to the other side and increase the bandwidth demands across the sub-trees. On the other hand, however, we can increase the number of co-located APPs after exchanging VMs in the left and right sub-trees, which will greatly reduce the intra-application bandwidth requirement. Let C_1 represent the increased portion of bandwidth due to the movement of APP VMs, and C_2 denote the decreased portion of bandwidth caused by increasing the number of co-located APP VMs. If $C_1 > C_2$, we say the cost of increased portion is larger than the portion of bandwidth reduction, which is not recommended in our design. However, if $C_1 < C_2$, there is benefit by exchanging VMs since the total bandwidth cost achieves a reduction of $C_2 - C_1$.

For a more intuitive understanding of above results, we consider a simple example as shown in Fig. 6. The tenant request has been placed in two sub-trees proportionally according to the placement *principle 1* showed in Fig. 6(a). So the bandwidth requirements $B_{mb_k,mb_{k+1}}$ and $B_{mb,app}$ are reduced to zero since the traffic from MB_k can be fully processed by MB_{k+1}s or APPs in the same sub-tree. Note that the number of APPs in the left sub-tree is larger than the number of APPs in the right side. We then try to exchange the MB_2 in the left side and APP in the right side to reduce the traffic across the sub-trees between APPs.

The result of exchange is shown in Fig. 6(b). The dashed arrows show the change of traffic after exchange. The label associated with each arrow is the value of traffic change. The long-dashed arrows show the increased portion of the bandwidth in the two links. While the short-dashed arrow represents the decreased portion of bandwidth across the sub-trees. After the exchange, the MB_2 in the left side cannot fully handle the traffic from MB_1 in the same side. This leads to an extra flow from MB_1 in the left side to MB_2 in the right side expressed in the blue long-dashed arrow. Similarly, the APPs in the left side get additional capacity to process the extra flow from MB_2 in the right side due to the exchange showed by the green long-dashed arrow. For the intra-application communication, we get a decrease of bandwidth represented by the short-dashed arrow through increasing the number of co-located APPs in one side. Clearly, in this example, C_1 is equal to the sum of the two extra flows' bandwidth requirements computed as $b_2 + (b_2 + B^{ex}) = 2b_2 + B^{ex}$, where b_2 is the required capacity of MB_2 mentioned before. And C_2 is equal to $2B^{in}$ which is the bandwidth saving of intra-application communication.

Thus, if $C_2 > C_1$ or $2B^{in} > 2b_2 + B^{ex}$, it is a better choice to exchange, i.e., the total bandwidth consumption in the two links will be reduced if the decreased portion is larger than the increased portion of the bandwidth after the exchange. And, we can obtain the following exchange conditions for any MB_k using the same analyzing method as shown above:

$$\begin{aligned} 2B^{in} &> b_k + B^{ex}, k = 1; \\ 2B^{in} &> 2b_k + B^{ex}, k > 1. \end{aligned} \tag{3}$$

Bandwidth saving on the single link. We further analyze the bandwidth variation on the single link based on the exchanging strategy. For a better understanding, we consider the simple example shown in Fig. 6. We assume $b_2 = 200$, $B^{ex} = 600$, and $B^{in} = 502$ (in Mbps). It is clear that MB_2 satisfies the exchange condition mentioned above. The right link get a bandwidth reduction computed as $B^{in} - b_2 = 502 - 200 = 302$ Mbps after exchange. While the left link has a bandwidth increase of $(B^{ex} + b_2) - B^{in} = 600 + 200 - 502 = 298$ Mbps. So the total bandwidth has been reduced. But the exchange may lead to an unbalanced utilization of the two links. And the tenant request will be rejected if the remain bandwidth is smaller than 298 Mbps in the left link before the exchange. To avoid this situation, we reconsider the condition of exchanging to ensure that both links have bandwidth savings.

As shown in Fig. 6, the two links both get an extra traffic flow which causes a bandwidth increase (b_2 and $B^{ex} + b_2$ respectively) from the opposite side. In addition, there is a bandwidth reduction of B^{in} for intra-application communication in both of the links. Further, for the left link, there is a bandwidth variation $C_l = B^{ex} + b_2 - B^{in}$. While for the right link, the bandwidth variation is $C_r = b_2 - B^{in}$. If $C_l < 0$ (or $C_r < 0$), the bandwidth demand will be reduced in the left link (or right link) after exchange.

Hence, the condition to have bandwidth savings for both links is given as

$$C_l < 0 \quad and \quad C_r < 0 \tag{4}$$

i.e, if $B^{in} > (B^{ex} + b_2)$, the bandwidth will be saved for both of the links. Similarly, we can obtain the following

improved conditions to get a more appropriate exchanging for any middlebox MB_k.

$$\begin{aligned} &B^{in} > b_k \quad and \quad B^{in} > B^{ex}, k = 1; \\ &B^{in} > b_k + B^{ex}, k > 1. \end{aligned} \tag{5}$$

Further, using the revised exchangeable conditions to select the proper type of middleboxes for exchanging[5], we can achieve a bandwidth saving for the total bandwidth requirement across sub-trees while ensuring a bandwidth reduction for each of the links.

Algorithm 1 VM Placement Algorithm

```
1:  function PLACETENANT(Tenant t)
2:      st = FindlowestSubtree(t, null);
3:      while st do
4:          AllocateVM(t, t.mb, t.app, st);
5:          if ReserveBw(t) then
6:              return true;
7:          DeAllocate(t);
8:          st = FindLowestSubtree(t, st);
9:      return false;
10: function ALLOCATEVM(t, N_mb_app, st)
11: /* N_mb_app stores the numbers of instances for
12:     each type of middleboxes, and application VMs */
13:     if level(st)==0 then /*st is a server*/
14:         PlaceMBAPP(t, N_mb_app, st);
15:     else
16:         for each subtree v of st do
17:             if freeVM(v)>0 then
18:                 if freeVM(v)<TotalVM(N_mb_app) then
19:                     place MBs and APPs
20:                     proportionally under v;
21:                     /* N_v_mb_app stores the numbers of
22:                     MBs and APPs placed under v */
23:                     N_mb_app = N_mb_app − N_v_mb_app;
24:                     ExchangeMbApp(t,N_mb_app,N_v_mb_app);
25:                     AllocateVM(t, N_v_mb_app, v);
26:                 else
27:                     AllocateVM(t, N_mb_app, v);
```

D. VM Placement Algorithm

Based on above placement principles and exchangeable conditions, we design an effective heuristic VM placement algorithm that achieves bandwidth savings. The basic idea behind the algorithm is to place MBs and APPs proportionally initially to minimize the bandwidth consumption due to the unbalanced capacity assignment between middlebox and application VMs. And we then try to co-locate more APPs to reduce the bandwidth cost of intra-application communication through exchanging MBs and APPs.

[5]Eq. 5 is a sufficient but not necessary condition for bandwidth saving through exchanging, and we can also have bandwidth saving even the VMs are placed randomly initially by exchanging using Eq. 5.

Algorithm 2 Exchange Algorithm

```
1:  function EXCHANGEMBAPP(t,N_mb_app,N_v_mb_app)
2:      mb = GetExchangeableMB(t);
3:      if mb!=null then
4:          num is assigned as the number of types
5:          of the exchangeable MBs;
6:          k = 0;
7:          if N_mb_app[app] > N_v_mb_app[app] then
8:              while k < num and N_v_mb_app! = 0 do
9:                  if N_mb_app[mb[k]] == 0 then
10:                     k++; continue;
11:                 /* update the numbers of MBs
12:                 and APPs in both sides      */
13:                 N_mb_app[mb[k]]- -; N_mb_app[app]++;
14:                 N_v_mb_app[mb[k]]++; N_v_mb_app[app]- -;
15:         else
16:             while k < num and N_mb_app! = 0 do
17:                 if N_v_mb_app[mb[k]] == 0 then
18:                     k++; continue;
19:                 /* update the numbers of MBs
20:                 and APPs in both sides      */
21:                 N_v_mb_app[mb[k]]- -; N_v_mb_app[app]++;
22:                 N_mb_app[mb[k]]++; N_mb_app[app]- -;
```

Algorithm 1 depicts the process to handle a tenant request. The algorithm starts with $PlaceTenant$ (line 1), which takes a tenant request t as an input. To save the precious core-level bandwidth, we traverse the datacenter topology in a bottom-up manner to search for a valid lowest level sub-tree that has enough VM slots for t (line 2). For the candidate sub-tree, we place t in it with $AllocateVM$ (line 4) and then check whether the bandwidth requirement can be satisfied (line 5). If the trial fails due to the lack of bandwidth, $DeAllocate$ (line 7) is invoked to release the resources reserved for the tenant request. And the algorithm continues to find another sub-tree in the same level or upper level if there is no sub-tree in the same level (line 8).

$AllocateVM$ (line 10) is a recursive function, which takes a tenant request t, including its MB and APP VMs numbers and a sub-tree st as inputs. The function first checks if the st is the lowest level sub-tree or physical server: if so, it allocates corresponding number of slots for t according to the parameters of N_{mb_app} in the server and then returns. If st is a higher level switch, we then invoke $AllocateVM$ recursively for each sub-tree v of st which has free VM slots. For the sub-tree v that cannot provide enough slots for the unallocated MB and APP VMs, we first attempt to place appropriate number of MBs and APPs according to the placement *principle 1* to fill the subtree v and then co-locate as many APPs as possible by exchanging MBs and APPs using the exchangeable conditions of Eq. 5 . After that, we invoke the function $AllocateVM$ (line 25) for the adjusted MBs and APPs. While for the sub-tree of st that has sufficient slot resource, $AllocateVm$ is invoked directly (line 27).

In Algorithm 2, $ExchangeMbApp$ (line 1) is invoked to

exchange the positions of MBs and APPs. The function first identifies the types of middleboxes that can achieve bandwidth saving through exchanging their and APPs' positions by using the exchangeable conditions of Eq. 5 (line 2). If the exchangeable middleboxes exist, we then check the number of APPs in the two sub-trees and exchange the smaller one with the MBs to increase the degree of co-location for APPs. We execute the exchanging until all exchangeable MBs or APPs have been exchanged. If current exchangeable MBs have been exchanged completely, we then select the next exchangeable type of MBs to exchange (line 9). After each exchanging, we update the numbers of MBs and APPs in the two sides.

V. Middlebox Scaling Approach

To address the middlebox scaling problem, instead of allocating new resource of VMs and network after MBs are placed in datacenters, we introduce a probability model to represent the variable resource requirement of individual tenant. Through statistically multiplexing on VMs, we are able to balance the physical resource consumption and scalability requirement of software MBs.[6]

We re-examine the MB request and take the time-varying resource requirements of MBs into consideration. Specifically, the MBs' requirement is composed of a basic requirement and a scaling requirement with probability. Based on this model, we replace the $\langle T, M\rangle$ in Section III-B with tuple $\langle T, M, m, P\rangle$, where T and M denote the type of MBs and the corresponding number of VMs in the basic requirement, and m denotes the number of VMs in the scaling requirement with probability P. For example, given $T = \{\text{FW, IDS, LB}\}, M = \{2, 4, 2\}, m = \{1, 2, 1\}, P = \{0.3, 0.2, 0.3\}$, we then know that the tenant requests 2 firewall MBs and the request may increase to $2 + 1 = 3$ MBs with probability of 0.2. Similarly, 4 IDS MBs (resp. 2 LB MBs) are needed with probability 0.8 (resp. 0.7) and 6 IDS MBs (resp. 3 LB MBs) with a probability of 0.2 (resp. 0.3). In summary, a tenant's request can be specified by multiple virtual networks (VNs) and each VN can be specified by an eight-tuple $\langle N, B^{in}, B^{ex}, dependencies, T, M, m, P\rangle$. It allows elasticity by having a probability model for the number of MBs needed in each VN.

With regard to the probability model, we can provide few redundancy which provides tradeoff between utilization and availability. By sharing free VMs, a specific MB can be soon deployed on the free VM once the scale-up/out requirement comes. For simplicity, we reuse the notation T to indicate how many types of MBs are required by the tenant. Let X represent the number of middlebox instances actually scaled. Clearly, $X = \sum_{i=1}^{T} X_i \cdot m_i$, where X_i (taking values in $\{0, 1\}$) indicates whether the scaling request of MB_i occurs. Assuming the resource requirements of different middleboxes

[6]More details about the probability model and VM multiplexing can be found in our prior work [26].

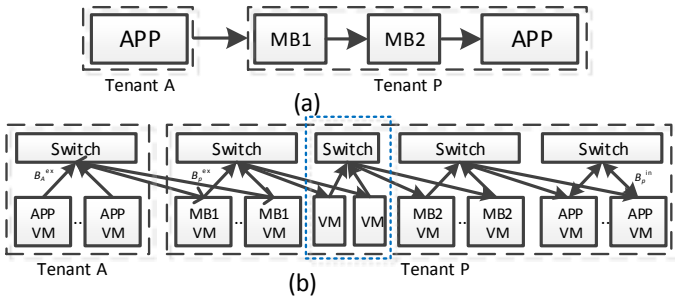


Fig. 7. Virtual Network Model. (a) the communication between two tenants; (b) cascade of TAG

are independent and v free VMs are reserved for multiplexing, the probability of the collision happening is

$$\begin{aligned} Pr\left[X > v\right] &= Pr\left[\sum_{i=1}^{T} X_i \cdot m_i > v\right] \\ &= Pr\left[X = v+1\right] + Pr\left[X = v+2\right] \\ &+ ... + Pr\left[X = m_p\right] \\ &= \sum_{i=v+1}^{m_p} Pr\left[X = i\right], \end{aligned} \tag{6}$$

where m_i represents the number of VMs for MB_i in the scaling requirement and m_p is the possible maximum number of scaled MB VMs and equal to the sum of all numbers of VMs in the scaling requirement, i.e., $m_p = \sum_{i=1}^{T} m_i$. By providing enough number of free VMs during virtual middleboxes network deployment, the probability of collision can be guaranteed to be less than a collision threshold ρ when the system is on board. The number of v reserved for sharing can be determined by solving the following function:

$$Pr\left[X > v\right] \leq \rho \quad or \quad Pr\left[\sum_{i=1}^{T} X_i \cdot m_i > v\right] \leq \rho. \tag{7}$$

The upper bound of collision probability $Pr\left[X > v\right]$ can be achieved based on the Markov's inequality, i.e.,

$$Pr\left[X > v\right] \leq \frac{E(X)}{v}. \tag{8}$$

$E(X)$ is the expectation of X and can be calculated by $E(X) = E(\sum_{i=1}^{T} X_i \cdot m_i) = \sum_{i=1}^{T} E(X_i) \cdot m_i$. Generally, let $\frac{E(X)}{v} = \rho$, we can obtain a rough evaluation for v. However, for a higher utilization of reserved VMs, a preciser value of v is required. Considering the numbers of MB types and possibly scaled VMs are small, we can easily compute the probabilities $Pr\left[X = i\right]$ related to the possible numbers of scaled VMs according to the tuple $\langle m, P\rangle$. Further, we can get the minimum number of extra VMs needed to be reserved by solving:

$$\min_{v \geq 0}(v) \quad s.t. \quad \sum_{i=v+1}^{m_p} Pr\left[X = i\right] \leq \rho. \tag{9}$$

Therefore, the total number of MB VMs to be allocated for one tenant is $|M|+v$, where $|M|$ is the number of basic MBs requirement and v is the minimum additional free VMs to be allocated as backup resource.

Fig. 7(b) uses the cascade of TAG model to abstract the communication dependency of VNs given in Fig. 7(a). In comparison with Fig. 4, the VMs inside the blue dotted line are pre-allocated VMs for middlebox scaling requirement, which are shared by the scaling requests of both types of middleboxes. Specifically, the reserved free VMs are pre-allocated under the same sub-tree as normal allocated VMs, which can greatly reduce unnecessary bandwidth consumption crossing racks or sub-trees. If there is scale-out or failover request, specified MBs will be placed on these additional VMs to provide performance guarantee.

VI. Evaluation

In this section, we first evaluate the benefits of our VN model with exchanging placement algorithm. Then we simulate the tenants' scaling requests and verify the effectiveness of our probability model based scaling approach.

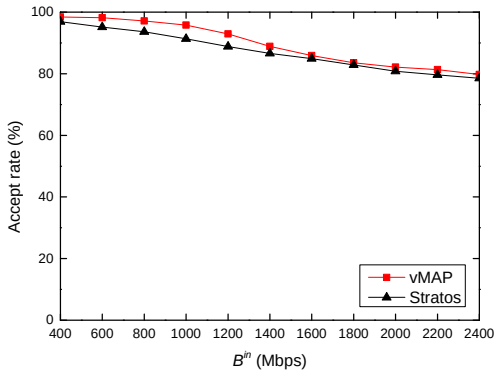
A. Simulation Setup

We build a tree-like three-layer network topology, which is inspired by a real cloud datacenter with 30000+ VMs. Each ToR switch is connected to 40 physical servers with 10Gbps NICs. Each PM has 8 slots. The oversubscription ratio is 4:1 at the leaf level and 10:1 at the spine layer.

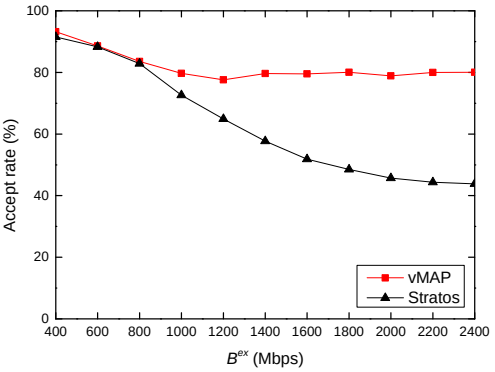
Tenant. The intra-application bandwidth B^{in} and inter-tenant bandwidth B^{ex} follow normal distribution. The maximal capacity b_k of each type of middleboxes follows the normal distribution with average value of 1000. For simplicity, we assume that each tenant request only contains one middlebox chain. The length of the middlebox chain is randomly selected in [1,5]. The number of APP VMs required corresponding to a middlebox chain follows uniform distribution in the range from 3 to 10. Besides, the fraction of open tenants is set to be 10% while the fraction of client tenants is defined as 25%. Furthermore, there are also some tenants that have communication request with each other. We set the fraction of these tenants to be 15%. The average number of communication dependencies is 2 [4].

VM placement. We consider two placement algorithms: 1) a VM placement strategy described in Stratos [1] which logically partitions the VMs into per-rack partitions and places them with minimal inter-partition communication; 2) our placement algorithm (vMAP) that places the MB and APP VMs carefully given in Algorithm 1. To compare the effectiveness of the two placement algorithms, we create tenant requests continuously until the datacenter could not accept tenant request any more.

Scaling requirement. For each tuple $\langle m, P\rangle$ of the scaling requirement, the scaling number m_i of each MB is equally distributed in [1,8] and the scaling probability P_i follows the normal distribution with a mean value of 0.5.



(a) Accept rates under different B^{in}



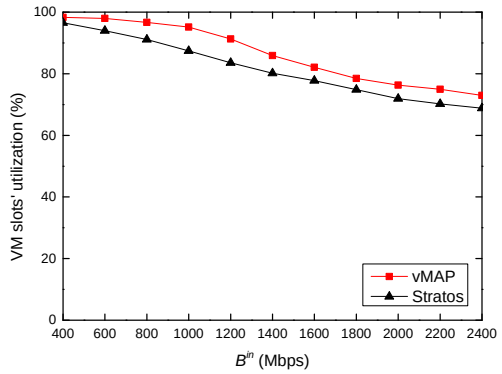
(b) Accept rates under different B^{ex}

Fig. 8. Accept rates under different B^{in} and B^{ex}

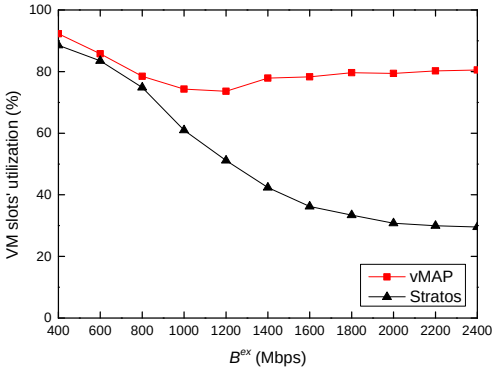
B. Simulation Results

Placement of middlebox and application. We first evaluate the tenant accept rate under different B^{in} and B^{ex} in Fig. 8. Fig .8(a) shows the accept rate of tenant requests with various intra-application communication demands (B^{in}). B^{in}=1200 means that the tenants' B^{in} follows the normal distribution with average value of 1200 and variation equal to B^{in}/3=400. In Fig. 8(a), we find that both algorithms have similar accept rates with different B^{in}.

The result of accept rate with different external bandwidth (B^{ex}) is shown in Fig. 8(b). Similarly, B^{ex}=1200 means that B^{ex} follows the normal distribution with the average value of 1200 and variation equal to B^{ex}/3=400. We find that vMAP ensures a high accept rate even with a large B^{ex}. While the accept rate of Stratos drops quickly when B^{ex} is larger than 800Mbps. This is because Stratos uses more bandwidth resource in core-level links. More specifically, with the increased demand for VM slots in the datacenter, the VMs of the tenant will be spread across more racks. The placement strategy used in Stratos is more likely to select the racks that reside across several sub-trees to accommodate the tenant request, which may cause a big waste of bandwidth in the

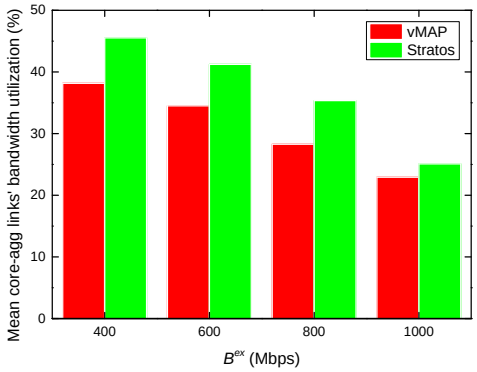


(a) VM slot utilizations under different B^{in}

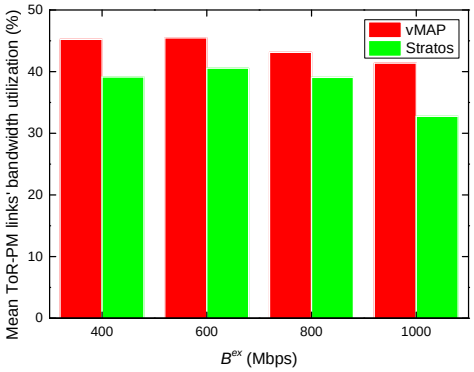


(b) VM slot utilizations under different B^{ex}

Fig. 9. VM slot utilizations under different B^{in} and B^{ex}



(a) Core-aggregation links' bandwidth utilizations



(b) ToR-PM links' bandwidth utilizations

Fig. 10. Bandwidth utilizations under different B^{ex}

core-level links. While our placement algorithm always tries to select the lowest-level sub-tree that has sufficient VM slots for the tenant, which can effectively save the limited bandwidth resource of the core-level links. By doing so, the proposed strategy vMAP allows more tenant requests to be accepted.

Fig. 9 shows the VM slot utilization of the datacenter under different B^{in} and B^{ex}. We can see that the results of VM slot utilization have similar changing trends compared to the accept rates shown in Fig. 8. This is reasonable since higher accept rates represent that more VM slots are allocated.

For a better understanding of above results, we further evaluate the downlink bandwidth utilization of both core switch-aggregation switch links and ToR-PM links. Note that the downstream traffic and upstream traffic appear at the same time. This leads to the result that the download and upload utilizations are in proportion. Hence, we just show the results of the download bandwidth here. The results are shown in Fig. 10, where the x-coordinate shows the intra-application bandwidth (B^{in}) consumption, and the y-coordinate shows the average bandwidth utilization of the core-aggregation and ToR-PM download links in the two subfigures respectively. In Fig. 10(a), we can see that vMAP has lower bandwidth

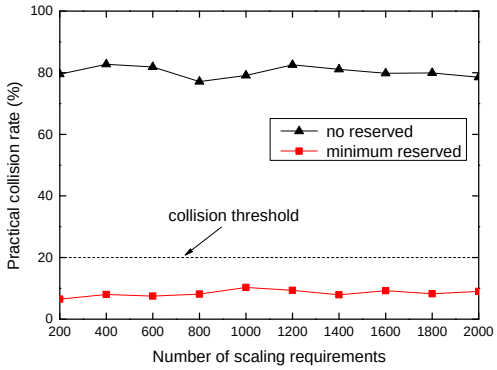


Fig. 11. Practical collision rates

utilization than Stratos in the core-aggregation links. On the other side, vMAP has a higher utilization of the PM-ToR links shown in Fig. 10(b) which, in turn, proves the effectiveness of lowest-level sub-trees locality placement of our algorithm. To sum up, vMAP can save the bandwidth efficiently for core-level links which are usually the bottlenecks for most

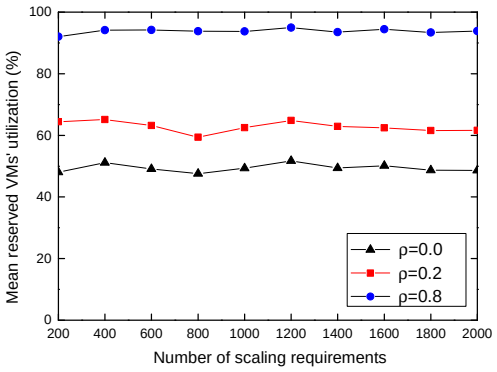


Fig. 12. Reserved VMs' utilizations

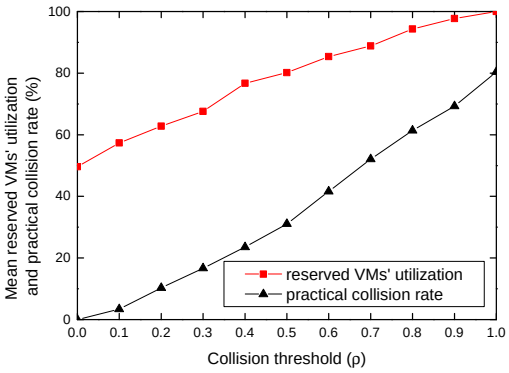


Fig. 13. Reserved VMs' utilizations and practical collision rates under different collision thresholds

oversubscription datacenter networks while making good use of the low-level bandwidth in the datacenter networks.

Middlebox scaling approach. We then evaluate the effectiveness of our probability model based scaling approach. Fig. 11 illustrates the collision rate with the increase of the number of scaling requirements. Specifically, we consider the collision rate of two different situations: 1) no free VMs reserved 2) and minimum number of free VMs needed to be reserved given a collision threshold $\rho = 0.2$. We can see that the collision rates are high if no free VMs are reserved. While the collision rate is ensured by reserving a few free VMs for the scaling requirements.

In Fig. 12, we evaluate the reserved VMs' utilization with different collision thresholds as the number of scaling requirements increases. As we can see the reserved VMs' utilizations perform steadily with the increase of the number of scaling requirements. And a larger collision threshold corresponds to a higher reserved VMs' utilization. In particular, we can guarantee that each scaling requirement is satisfied by setting a threshold $\rho = 0.0$ at the cost of wasting almost a half of reserved VMs resource, which is not economic. On the contrary, we can improve the reserved VMs' utilization by setting a large value of ρ (such as 0.8) and have to tolerate a high collision rate consequently.

To achieve a better tradeoff between the collision rate (performance) and the reserved VMs' utilization (cost) in practice, we further illustrate the correlations of the collision rate, reserved VMs' utilization, and collision threshold in Fig. 13 to guide the selection of the proper collision threshold. As we can see, the reserved VMs' utilization and collision rate grow linearly in terms of the collision threshold. When the collision threshold is equal to 0.4, we can have a higher reserved VMs' utilization ($\sim$80%) with a lower collision rate ($\sim$23%).

Summary. The proposed exchanging algorithm for middlebox and application can save the core-level bandwidth efficiently while making best use of the low-level bandwidth, which leads to a high accept rate of the tenant requests. And through VM resource reserving and multiplexing based on a probability model, we can satisfy the scaling requirement effectively while minimizing unnecessary resource waste.

VII. Conclusion

Since middleboxes are widely used in current networks, it is essential to consider middleboxes when deploying the tenant request with performance guarantee. In this paper, we introduced vMAP, a joint virtual middlebox and application deployment solution. We first model the tenant's requirement of software middleboxes and applications by augumenting software middleboxes as VMs in the Tenant Application Graph. A VM placement algorithm is then proposed to place these VMs (including both virtual middlebox and application VMs) in the underlying infrastructure. To address the middlebox scaling problem, we propose a probability model based approach to accommodating the middlebox scaling request by reserving a few additional free VMs, which is beneficial for fast deployment and saving precious network resource. The evaluation results have shown that the proposed joint virtual middlebox and application placement algorithm can efficiently use the resource of multi-tenant datacenters and provide performance guarantee for tenants with scaling requirement.

References

References

[1] A. Gember, A. Akella, A. Anand, T. Benson, R. Grandl, Stratos: Virtual middleboxes as first-class entities, Tech. Rep. TR1771, University of Wisconsin-Madison, 2012.

[2] A. Gember, R. Grandl, J. Khalid, A. Akella, Design and implementation of a framework for software-defined middlebox networking, in: Proceedings of the ACM SIGCOMM, ACM, 2013, pp. 467–468.

[3] H. Ballani, P. Costa, T. Karagiannis, A. Rowstron, Towards predictable datacenter networks, in: ACM SIGCOMM Computer Communication Review, Vol. 41, ACM, 2011, pp. 242–253.

[4] H. Ballani, K. Jang, T. Karagiannis, C. Kim, D. Gunawardena, G. O'Shea, Chatty tenants and the cloud network sharing problem, in: Proceedings of the USENIX NSDI, USENIX Association, 2013, pp. 171–184.

[5] A. Williamson, Has amazon EC2 become over subscribed, http://alan.blog-city.com/has amazon ec2 become over subscribed.htm.

[6] M. Al-Fares, A. Loukissas, A. Vahdat, A scalable, commodity data center network architecture, in: ACM SIGCOMM Computer Communication Review, Vol. 38, ACM, 2008, pp. 63–74.

[7] P. Patel, D. Bansal, L. Yuan, A. Murthy, A. Greenberg, D. A. Maltz, R. Kern, H. Kumar, M. Zikos, H. Wu, et al., Ananta: Cloud scale load balancing, in: ACM SIGCOMM Computer Communication Review, Vol. 43, ACM, 2013, pp. 207–218.

[8] J. Martins, M. Ahmed, C. Raiciu, V. Olteanu, M. Honda, R. Bifulco, F. Huici, Clickos and the art of network function virtualization, in: Proceedings of the USENIX NSDI, USENIX Association, 2014, pp. 459–473.

[9] R. Yu, G. Xue, V. T. Kilari, X. Zhang, Network function virtualization in the multi-tenant cloud, in: IEEE Network, Vol. 29, 2015, pp. 42–47.

[10] ISG web portal, Network Functions Virtualisation., in: http://portal.etsi.org/portal/server.pt/community/NFV/367, 2013.

[11] M. Dobrescu, K. Argyraki, S. Ratnasamy, Toward predictable performance in software packet-processing platforms, in: Proceedings of the USENIX NSDI, USENIX Association, 2012, pp. 11–11.

[12] V. Sekar, N. Egi, S. Ratnasamy, M. K. Reiter, G. Shi, Design and implementation of a consolidated middlebox architecture, in: Proceedings of the USENIX NSDI, USENIX Association, 2012, pp. 24–24.

[13] K. Benzekki, A. El Fergougui, A. Elbelrhiti Elalaoui, Software-defined networking (SDN): a survey, in: Security and Communication Networks, Vol. 9, Wiley Online Library, 2016, pp. 5803–5833.

[14] Z. A. Qazi, C.-C. Tu, L. Chiang, R. Miao, V. Sekar, M. Yu, Simplefying middlebox policy enforcement using SDN, in: ACM SIGCOMM computer communication review, Vol. 43, ACM, 2013, pp. 27–38.

[15] B. Anwer, T. Benson, N. Feamster, D. Levin, J. Rexford, A slick control plane for network middleboxes, in: Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking, ACM, 2013, pp. 147–148.

[16] A. Gember-Jacobson, R. Viswanathan, C. Prakash, R. Grandl, J. Khalid, S. Das, A. Akella, OpenNF: Enabling innovation in network function control, in: ACM SIGCOMM Computer Communication Review, Vol. 44, ACM, 2014, pp. 163–174.

[17] L. Popa, G. Kumar, M. Chowdhury, A. Krishnamurthy, S. Ratnasamy, I. Stoica, Faircloud: Sharing the network in cloud computing, in: Proceedings of the ACM SIGCOMM, ACM, 2012, pp. 187–198.

[18] L. Popa, P. Yalagandula, S. Banerjee, J. C. Mogul, Y. Turner, J. R. Santos, Elasticswitch: Practical work-conserving bandwidth guarantees for cloud computing, in: ACM SIGCOMM Computer Communication Review, Vol. 43, ACM, 2013, pp. 351–362.

[19] R. Mijumbi, J. Serrat, J.-L. Gorricho, N. Bouten, F. De Turck, R. Boutaba, Network function virtualization: State-of-the-art and research challenges, in: IEEE Communications Surveys & Tutorials, Vol. 18, IEEE, 2016, pp. 236–262.

[20] Huawei NFV Open Lab, in: http://pr.huawei.com/en/news, 2015.

[21] CloudNFV, in: http://www.cloudnfv.com/, 2015.

[22] S. Mehraghdam, M. Keller, H. Karl, Specifying and placing chains of virtual network functions, in: IEEE International Conference on Cloud Networking, 2014, pp. 7–13.

[23] R. Cohen, L. Lewin-Eytan, J. S. Naor, D. Raz, Near optimal placement of virtual network functions, in: Proceedings of the IEEE INFOCOM, 2015, pp. 1346–1354.

[24] J. Lee, Y. Turner, M. Lee, L. Popa, S. Banerjee, J.-M. Kang, P. Sharma, Application-driven bandwidth guarantees in datacenters, in: Proceedings of the ACM SIGCOMM, ACM, 2014, pp. 467–478.

[25] N. G. Duffield, P. Goyal, A. Greenberg, P. Mishra, K. K. Ramakrishnan, J. E. van der Merive, A flexible model for resource management in virtual private networks, in: ACM SIGCOMM Computer Communication Review, Vol. 29, ACM, 1999, pp. 95–108.

[26] S. Zhang, Z. Qian, Z. Luo, J. Wu, S. Lu, Burstiness-aware resource reservation for server consolidation in computing clouds, in: IEEE Transactions on Parallel and Distributed Systems, Vol. 27, IEEE, 2016, pp. 964–977.

Jian Wang received his Bachelor degree from the Department of Computer Science and Technology, Harbin Engineering University. He is currently a third year graduate student of the Department of Computer Science and Technology, Nanjing University. He currently focuses on the field of resource management in datacenter networks.

Xuewei Zhang received her Bachelor degree and Master degree from the Department of Computer Science and Technology of Nanjing University in 2014 and 2017, respectively. Her research interests include datacenter network system and scheduling algorithm.

Xiaoliang Wang received his Ph.D degree from Graduate School of Information Sciences, Tohoku University, Japan. He is currently an Associate Professor in the Department of Computer Science and Technology, Nanjing University, China. He was with the same university as an Associate Professor, from 2010-2014. His research interests include optical switching networks, scheduling algorithm and data center network system. He has published over 30 technical papers at premium international journals and conferences, including IEEE TIT, IEEE TCOM, IEEE INFOCOM, USENIX ATC, USENIX FAST, etc.

Sheng Zhang received his BS and PhD degrees from Nanjing University in 2008 and 2014, respectively. Currently, he is an assistant professor in the Department of Computer Science and Technology, Nanjing University. He is also a member of the State Key Lab. for Novel Software Technology. His research interests include cloud computing and mobile networks. To date, he has published more than 40 papers, including those appeared in *IEEE TMC*, *IEEE TPDS*, *IEEE TC*, ACM MobiHoc, IEEE ICDCS, and IEEE INFOCOM. He received the Best Paper Runner-Up Award from IEEE MASS 2012.

Sanglu Lu received the BS, MS, and PhD degrees from Nanjing University in 1992, 1995, and 1997, respectively, all in computer science. She is currently a professor in the Department of Computer Science and Technology and the State Key Laboratory for Novel Software Technology. Her research interests include distributed computing, wireless networks, and pervasive computing. She has published more than 80 papers in referred journals and conferences in the above areas. She is a member of the IEEE.