Modeling and Deploying Networklet

Sheng Zhang[†], Yu Liang[†], Zhuzhong Qian[†], Mingjun Xiao[§], Jie Wu[‡], Fanyu Kong[♯], and Sanglu Lu[†]
[†]State Key Lab. for Novel Software Technology, Nanjing University, P.R. China
[§]School of Computer Science and Technology / Suzhou Institute for Advanced Study,
University of Science and Technology of China, P.R. China
[‡]Center for Networked Computing, Temple University
[♯]Ant Financial, P.R. China
Email: sheng@nju.edu.cn

***Abstract*—The lines between IaaS, PaaS, and SaaS are becoming blurred as datacenter providers seek to create cloud platforms that can widen their appeal to developers. With this kind of hybrid datacenter, resource requests from tenants are increasingly transforming into hybrid requests that may simultaneously demand IaaS, PaaS, and SaaS resources. This paper tackles the challenge of modeling and deploying hybrid tenant requests in datacenter networks, for which we coin "networklet" to represent a set of VMs that collaboratively provide some PaaS or SaaS service. Through extracting networklets from tenant requests and thus sharing them between multiple tenants, we can achieve a win-win situation for datacenter providers and tenants. Extensive evaluations show that, the proposed model and deployment algorithm indeed improve DCN resource utilization while maintaining performance guarantee.**

I. Introduction

A. Motivation

Today's public datacenters (*e.g.*, Amazon EC2, and Microsoft Azure) focus on computation-oriented resource reservation [11, 19], which only allows tenants to specify computing and memory demands, but totally ignores networking, *i.e.*, most datacenters just offer best-effort networking service. Although simple, this model results in highly unpredictable performance of tenants virtual machines (VMs) [12].

To provide performance guarantee, prior works [12, 13, 15–17, 19, 20] have proposed several novel abstractions that allow tenants to explicitly specify networking as well as computing demands. However, most of them fit comfortably under one of two headings: hose [12, 15, 19], or clique [13, 16–18, 20, 21, 23, 24]. In the hose model, all tenant VMs are connected to a common virtual switch by links of homogeneous or heterogenous capacities; while in the clique model, tenants can specify bandwidth requirements between all pairs of VMs. In fact, these two types of abstractions represent two extremes in the design space; we want to propose a new abstraction model that may help datacenter providers cater to resource allocation in hybrid datacenters.

We find that, the lines between Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS) are becoming blurred as datacenter providers seek to create cloud platforms that can satisfy the needs of enterprises and widen their appeal to developers. According to some Forrester's top cloud analysts [10], the popular wisdom that cloud computing comes in three flavors—IaaS, PaaS, and SaaS—no longer describes reality. Specifically, IaaS providers are trying to cater to application development (*e.g.*, AWS Elastic Beanstalk provides PaaS-like development layers [5]); PaaS providers are exposing more infrastructure APIs to tenants (*e.g.*, Engine Yard begin to support some IaaS APIs [6]); and SaaS providers are allowing tenants to set up their apps within their software (*e.g.*, Salesforce is providing application extension tools [9]). With this kind of hybrid datacenter, resource requests from tenants are increasingly transforming into hybrid requests that may simultaneously demand IaaS, PaaS, and SaaS resources (*e.g.*, a tenant can set up a website using IaaS VMs, and meanwhile store/retrieve pictures of the website using SaaS APIs [8]).

B. Proposed Approach and Key Contributions

Motivated by these observations, this paper tackles the challenge of modeling and deploying hybrid tenant resource requests in hybrid datacenter networks (DCNs). A hybrid tenant request can be seen as a set of IaaS VMs, the bandwidth requirements between them, and a set of PaaS or SaaS services it may access. We coin the word *networklet* [22] to represent a set of VMs that collaboratively provide some PaaS or SaaS service. As we know, tenants usually do not access or occupy a PaaS or SaaS service throughout the duration of the request; thus, it is reasonable to share networklets among multiple tenants. As long as the sum of the demands for a networklet from multiple tenants does not exceed the service capacity of a networklet, we can guarantee predictable performance of tenant applications. We will shortly see that, extracting networklets from tenant requests greatly benefit both tenants and providers, achieving a win-win situation.

The primary contribution of this paper is the concept of networklet. A networklet can be shared by multiple tenant requests, depending on its service capacity and tenant demands for it. From the perspective of a tenant, it pays less than before as it shares some networklets with others. From the perspective of a provider, its revenue is higher than before as physical resource utilization becomes higher.

The second contribution is the deployment algorithm for hybrid tenant requests with shared networklets. We want to use a simple yet efficient algorithm to solve the deployment problem, after observing many practical algorithms and simple

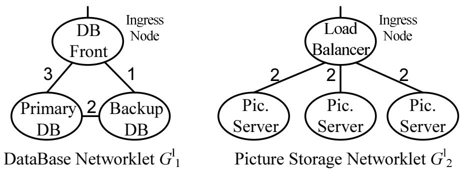


Fig. 1: Two networklet examples that represent database and picture storage services, respectively.

principles behind them implicitly advocated by many pioneers, *e.g.*, [12, 17, 19, 20]. The main idea is to put VMs that have large bandwidth demands between them as close as possible, so as to reduce the bandwidth consumption.

Thirdly, extensive evaluations are conducted to confirm the effectiveness and advantages of the proposed concept and algorithm. For a batch of tenant resource requests, the proposed algorithm achieves a completion time that is only 64% of baseline's completion time, and of course, produces a better physical resource utilization than two baselines.

II. Networklet and Hybrid Tenant Request

A. Networklet

We coin networklet from the standpoint of IaaS datacenter providers, who are in a race to differentiate themselves by producing tenant-friendly features that may increase the agility of tenants. Almost all IaaS providers have focused on computing and storage resources in the past. Now more and more of them begin to pay attention to services that can accelerate the development by abstracting away the complexities of the underlying infrastructure such as orchestration and automation. For example, Amazon starts to offer four different databases as a service (DBaaS) offerings: RDS for MySQL/Oracle [3], DynamoDB for NoSQL [1], ElasticCache for in-memory caching service [2], and Redshift for cost-effective data warehousing [4]; and OpenStack also includes its DBaaS component, *i.e.*, Trove [7], in its latest release.

Therefore, *we use networklet to represent a PaaS or SaaS service while taking a stand as IaaS providers*. Fig. 1 shows two networklet examples: database networklet—a PaaS service, and picture storage networklet—a SaaS service. A networklet is denoted by an undirected graph, where vertices represent VMs and edges represent bandwidth requirements between them. For ease of exposition, we choose to abstract away details of the non-network resources as in previous work [12, 19]; thus, each VM requires a fixed VM slot in DCN servers. Every networklet has an ingress node which is the first node accessed by tenant requests. Each networklet has a service capacity, indicating how much workload it can handle in time.

More formally, we assume that the IaaS provider decides to offer L types of networklets, based on market investigation and historical data mining. The j-th networklet is denoted by a graph $G^l_j = (V^l_j, E^l_j)$, where V^l_j is the set of VMs and E^l_j is the set of bandwidth requirements. For an edge $e^l = (v^l_a, v^l_b) \in E^l_j$, $b(e^l)$ or $b(v^l_a, v^l_b)$ denotes its bandwidth requirement. G^l_j has a service capacity C_i. Without loss of generality, we assume v^l_0 is its ingress component/node.

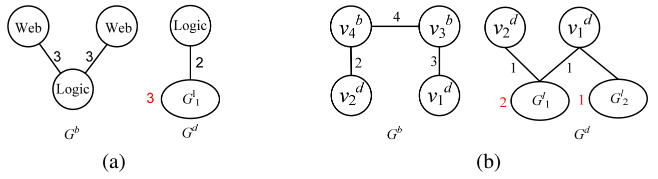


Fig. 2: HTR examples, where G^l_1 and G^l_2 denote the database and picture storage networklets, respectively, in Fig. 1. Note that, G^d is a bipartite graph specifying the connections between IaaS VMs and networklets.

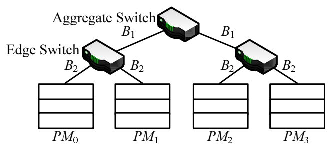


Fig. 3: A tree-like DCN example with $K = 2$, $D = 2$, and $Q = 3$

B. Hybrid Tenant Request (HTR)

When an IaaS datacenter specifies L types of networklets, resource requests from tenants could be hybrid.

A hybrid tenant request (HTR) contains a set of IaaS VMs, the bandwidth requirements between them, and a set of networklets it access. We choose to use two graphs, *i.e.*, $G^b = (V^b, E^b)$, and $G^d = (V^d, U^d, E^d)$, to represent an HTR. The basic graph G^b specifies IaaS VMs and the bandwidth requirements between them. For an edge $e^b = (v^b_1, v^b_2) \in E^b$, $b(e^b)$ or $b(v^b_1, v^b_2)$ denotes its bandwidth requirement. The external graph G^d tells us its requirements on networklets. G^d is a bipartite graph with two non-overlapping vertex set V^d and U^d, where V^d is a non-empty subset of V^b and U^d is the set of networklets it access. For an edge $e^d = (v^d, u^d) \in E^d$, $b(e^d)$ or $b(v^d, u^d)$ denotes its bandwidth requirement. Besides, each vertex u^d in U^d is associated with a weight $c(u^d)$, which is the demand for the service capacity of the corresponding networklet. Each HTR has a lifetime of t, *i.e.*, after the HTR is successfully deployed for a duration of t, it is finished and the physical resources allocated to it would be released.

The principle behind the notations in this paper is that, superscripts "l", "b", and "d" represent networklet, basic graph, and external graph, respectively.

In Fig. 2(a), the number next to G^l_1 indicates that, the demand for the service capacity of G^l_1 is 3. Fig. 2(b) shows a request that requires access to two networklets G^l_1 and G^l_2.

C. Datacenter Network

We assume the DCN topology is a simple tree[1], however, extending our work to other topologies is not hard. More specifically, denote by T the underlying DCN, which is a full K-ary tree of depth D. Thus, there are altogether $M = K^D$ physical machines (PMs). Each PM contains Q VM slots, *i.e.*, each PM could host at most Q tenant or networklet VMs. The bandwidth capacity of the physical links in the same layer are assumed to be same; denote by B_i the bandwidth capacity of the links between $(i-1)$-th and i-th layer switches. Fig. 3

[1]Our private conversation with researchers from two large cloud vendors shows that, the topologies of their datacenters are indeed simple trees.

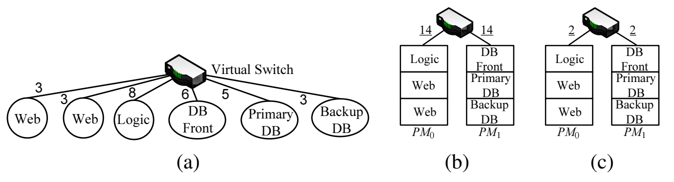


Fig. 4: Advantages of HTR+networklet over hose. The underlined number next to each edge represents the bandwidth used in each corresponding link. (a) The hose model. (b) Deploying the hose model in Fig. 4(a). (c) Deploying HTR+networklet in Fig. 2(a).

shows an example where $K = 2$, $D = 2$, and $Q = 3$. Without loss of generality, we denote DCN PMs in the figure from left to right by PM_0, PM_1, ..., and PM_{M-1}.

III. Advantages of HTR+Networklet

Comparison with the hose model. Fig. 4(a) shows the corresponding hose model of the same tenant request shown in Fig. 2(a). When converting the request in Fig. 2(a) into the one in Fig. 4(a), we must ensure that, each tenant VM has enough bandwidth for communication with the other VMs. For example, the maximum rate of the 'Logic' VM exchanging data is 3+3+2=8 in Fig. 2(a), so the virtual link connecting the 'Logic' VM to the virtual switch is 8 in Fig. 4(a).

Fig. 4(b) shows the deployment of the hose model in the DCN in Fig. 3, where the underlined number next to each edge represents the bandwidth used in each corresponding link. No matter how to partition these six tenant VMs, the total physical bandwidth it consumes is 14+14=28. However, if we use the HTR+networklet model, the bandwidth it consumes is only 2+2=4, as demonstrated in Fig. 4(c). We see that, HTR+networklet better resembles physical topologies and thus conserves physical bandwidth consumption.

Comparison with the clique model. Fig. 5(a) shows the corresponding clique model of the same tenant request shown in Fig. 2(a). The only difference between Fig. 5(a) and Fig. 2(a) is that, the database networklet is shared in Fig. 2(a) while it is exclusively occupied by the 3-tiered web application in Fig. 5(a). For comparison purpose, Figs. 5(b) and 5(c) show another tenant request that needs two VMs (VM1 and VM2) plus the database networklet.

Fig. 5(d) shows the deployment of the two tenant requests using the clique model in Figs. 5(a) and 5(b), where the underlying DCN is shown in Fig. 3. We find that, they consume altogether 11 VM slots. However, suppose the service capacity of the database networklet is 10, which is larger than the sum of the requirements of these two tenant requests, *i.e.*, $3 + 4 < 10$; then they can share the database networklet. The resulting deployment is shown in Fig. 5(e), which consumes altogether 8 VM slots. We see that, HTR+networklet achieves a better computing resource utilization through sharing networklets.

Summary. Table I summarizes the comparison results of hose, clique, and HTR+networklet on three design dimensions. Overall, our HTR+networklet model not only closely resembles the physical topologies used by datacenter tenants, but also improves physical resource utilization.

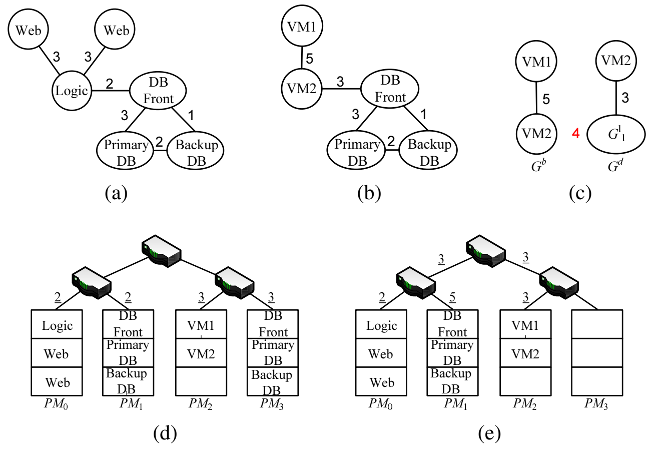


Fig. 5: Advantages of HTR+networklet over clique. (a) The clique model of the request in Fig. 2(a). (b) and (c) Another request in the clique model and the HTR+networklet model, respectively. (d) Deploying two clique-based requests in Figs. 5(a) and 5(b). (e) Deploying two HTRs in Figs. 2(a) and 5(b).

TABLE I: Comparison between the hose, clique, and HTR+networklet models

Abstraction	Tenant Cost	Provider Revenue	Provider Flexibility
VC/VOC [12]	High	Low	High
VDC [16]/VN [20]	Medium	Medium	Low
HTR+Networklet	Low	High	Medium

IV. Deploying HTR+Networklet

A. Overview

We concentrate on the online version of the deployment problem. HTRs arrive one by one over time, we want to design an algorithm to allocate resources for an HTR.

The ultimate goal of deployment is to maximize provider revenue while guaranteeing tenant application performance. As we mentioned before, this goal is equivalent to maximizing physical resource utilization, since DCN usually charges a tenant based on the amount of resources reserved for it. Maximizing resource utilization is further reduced to conserving physical resource consumption.

To conserve physical resource consumption, we have two sorts of strategies. The first one is to share networklets among multiple tenants, after observing a tenant usually does not occupy networklets throughout the duration of the HTR. As long as the sum of the demands for a networklet from multiple tenants does not exceed the service capacity of a networklet, we can guarantee predictable performance of tenant applications. The second one is to place VMs that have large bandwidth requirements between them as close as possible, so as to reduce the bandwidth consumption in underlying DCNs.

B. Preliminaries

Denote by P the set of all PMs in the underlying DCN. For a tenant VM v, we use $h(v)$ to represent the location of it; we also use $h(S)$ to denote the locations of a set S of VMs.

- The function $r(\cdot)$ gives the amount of residual resources of an entity. For example, $r(PM_i)$ denotes the amount of available VM slots on PM_i; $r(G^l)$ denotes the residual service capacity of a networklet G^l.

- The distance $d(\cdot,\cdot)$ returns the number of hops between two entities, *e.g.*, $d(PM_i, PM_j)$ is the number of hops between PM_i and PM_j; $d(v_i, v_j)$ is the number of hops between two PMs that host v_i and v_j, respectively.
- The function sort$(S, metric, +/-)$ sorts all elements in a set S in the non-decreasing (+) or non-increasing ($-$) order of the given $metric$. For example, sort$(P, r(PM), -)$ sorts all PMs in the non-increasing order of the residual resource of each PM; sort$(P, d(PM, PM_0), +)$ sorts all PMs in the non-decreasing order of the distance between each PM and PM_0.

Algorithm 1 HTR+Networklet Deployment Alg. (HNDA)

```
Input: An HTR described by $G^b = (V^b, E^b)$ and $G^d = (V^d, U^d, E^d)$, a DCN described by $T$
Output: Allocation for this HTR
1: for i=1 to $|U^d|$ do
2:     AllocNL$(T, G^l_i, c(G^l_i))$
3:     $r(G^l_i) \leftarrow r(G^l_i) - c(G^l_i)$
4: end for
5: AllocHTR$(T, G^b, G^d)$
```

C. HTR+Networklet Deployment Algorithm

The HTR+Networklet Deployment Algorithm (HNDA) is shown in Alg. 1. It contains two parts: allocation for networklets (lines 1-4), and allocation for HTR (line 5).

Allocation for Networklets. As we mentioned before, an HTR is composed of G^b, which captures the relationships between its IaaS VMs, and G^d, a bipartite graph indicating the networklets it needs to access. For each networklet G^l it needs to access, if the underlying DCN has one instance of G^l with sufficient residual service capacity, *i.e.*, $r(G^l) > c(G^l)$ (line 1 of Alg. 2), where $c(G^l)$ is the demand for this networklet, then we do not need to set up a new one.

Otherwise, we must allocate resource for building a new instance of G^l. Lines 2-5 of Alg. 2 allocate VM slot for the ingress node v^l_0 of G^l, in which we first sort all PMs in the non-increasing order of their residual VM slots (line 3), and attempt to deploy v^l_0 in a PM with sufficient residual resource in the sorted order.

The rest of Alg. 2 deals with the other nodes in V^l. Denote by S the set of already-allocated nodes in V^l. In each iteration, we select an edge which has the maximum bandwidth requirement connecting an unallocated node v^l_1 and an already-allocated node v^l_2 (line 8); then we sort all PMs in the non-decreasing order of their distances to the host of v^l_2, *i.e.*, $h(v^l_2)$, and attempt to deploy v^l_1 in a PM with sufficient residual resource in the sorted order, after which we allocate bandwidth for this edge and update S.

Please note that, for a certain type of networklet, there may be multiple instances in the DCN, depending on the demands for a networklet.

Allocation for HTR. After the allocation for possible networklets is completed, we begin to allocate physical resources for the HTR itself. The IaaS VMs can be partitioned into two non-overlapped sets: V^d and $V^b - V^d$, where "$-$" represents the set minus operation. We first deal with nodes in V^d (lines 1-10 in Alg. 3), then deal with the rest nodes in V^b (lines 11-19 in Alg. 3).

Algorithm 2 AllocNL(T, G^l, c)

```
1: if $G^l$ with sufficient residual service capacity ($r(G^l) \geq c$) exists then return
2: // allocation for the ingress node $v^l_0$
3: sort$(P, r(PM), -)$
4: attempt to deploy $v^l_0$ in a PM with sufficient residual resource in the sorted order, update $T$
5: // allocation for the rest nodes in $V^l$
6: $S \leftarrow \{v^l_0\}$
7: while $V^l \neq S$ do
8:     $(v^l_1, v^l_2) \leftarrow \arg\max_{v^l_1 \in V^l - S, v^l_2 \in S} b(v^l_1, v^l_2)$
9:     sort$(P, d(PM, h(v^l_2)), +)$
10:    attempt to deploy $v^l_1$ in a PM with sufficient residual resource in the sorted order, update $T$
11:    allocate bandwidth for all edges between $v^l_1$ and $S$
12:    $S \leftarrow S \cup \{v^l_1\}$
13: end while
14: return
```

Denote by $V^{d\prime}$ the set of already-allocated nodes in V^d. In each iteration, we select an edge which has the maximum bandwidth requirement connecting an already-allocated networklet u^d and an unallocated node $v^d \in V^d - V^{d\prime}$ (line 4); then we sort all PMs in the non-decreasing order of their distance to the host of the ingress node of u^d, and attempt to deploy v^d in a PM with sufficient residual resource in the sorted order, after which we allocate bandwidth for all edges between v^d, U^d, and $V^{d\prime}$, and update $V^{d\prime}$. Using the same heuristic, we can allocate resources for the rest of V^b.

In retrospect, HNDA can be decomposed into four main phases: find a physical location for the ingress node of a networklet (lines 2-5 of Alg. 2), find physical locations for the rest nodes of a networklet (lines 5-13 of Alg. 2), find physical locations for nodes in the external graph (lines 1-10 of Alg. 3), and find physical locations for the rest nodes in the basic graph (lines 11-19 of Alg. 3).

D. Discussions

DCN Topology. A fat-tree network with more than 2 tiers of switches can be recursively yet easily constructed using 2 tier fat-tree networks as building blocks [15], and any results on 2 tier fat-tree networks can be extended to fat-tree networks with more tiers. Therefore, it is sufficient to show how to handle path diversity in the basic 2 tier fat-tree network: there are m core switches, each is connected with r edge switches, each of which is further connected to n servers. Thus, there are in fact m different paths between two edge switches. When applying

HNDA, we can take these m paths as a single path and use ECMP to balance packets among these paths.

Heterogeneous VM Demands. It is not hard to extend HNDA to respect heterogenous VM demands. When we place VMs on PMs, we just have to additionally check whether the residual resource is sufficient for the requirement of the current VM.

Algorithm 3 AllocHTR(T, G^b, G^d)

```
1: // allocation for nodes in $V^d$
2: $V^{d\prime} \leftarrow \emptyset$
3: while $V^d \neq V^{d\prime}$ do
4:     $e^d = (v^d, u^d) \leftarrow \arg \max_{v^d \in V^d - V^{d\prime}, u^d \in U^d} b(v^d, u^d)$
5:     sort$(P, d(PM, h(u^d)), +)$
6:     attempt to deploy $v^d$ in a PM with sufficient residual
       resource in the sorted order, update $T$
7:     allocate bandwidth for all edges between $v^d$ and $U^d$
8:     allocate bandwidth for all edges between $v^d$ and $V^{d\prime}$
9:     $V^{d\prime} \leftarrow V^{d\prime} \cup \{v^d\}$
10: end while
11: // allocation for nodes in $V^b - V^d$
12: $V^{b\prime} \leftarrow V^d$
13: while $V^b \neq V^{b\prime}$ do
14:     $e^b = (v_1^b, v_2^b) \leftarrow \arg \max_{v_1^b \in V^b - V^{b\prime}, v_2^b \in V^{b\prime}} b(v_1^b, v_2^b)$
15:     sort$(P, d(PM, h(v_2^b)), +)$
16:     attempt to deploy $v_1^b$ in a PM with sufficient residual
        resource in the sorted order, update $T$
17:     allocate bandwidth for edges between $v_1^b$ and $V^{b\prime}$
18:     $V^{b\prime} \leftarrow V^{b\prime} \cup \{v_1^b\}$
19: end while
20: return
```

Work-conserving. The bandwidth requirement serves as the minimum guarantee (min-guarantee) for tenants. Whenever there is any residual bandwidth resource, it could be fairly shared among coexisted flows/connections.

Generalized Networklet. Although networklet is coined to represent some PaaS or SaaS service. It can be generalized to more entities, *e.g.*, network virtualization functions (NFVs). Placing NFVs in DCNs is an interesting problem [14].

Service Model of Networklet. We assume each networklet has a service capacity and each HTR has a demand, and as far as the sum of demands is not larger than the service capacity, the tenant application performance is not affected. In fact, such kind of simplification is made for ease of exposition. We can easily extend the main idea of the paper to the scenario where the service model of a networklet follows queuing theory.

V. Performance Evaluation

A. Simulation Setup

In our simulations, the underlying multi-tenant datacenter topology is a simple tree with $K = 5$, $D = 3$, $Q = 10$, $B_1 = 60$, and $B_2 = 120$ (see Fig. 3). We assume that there are $L = 5$ types of networklets, and their topologies are shown in Fig. 6. We leave as future work extensive evaluations

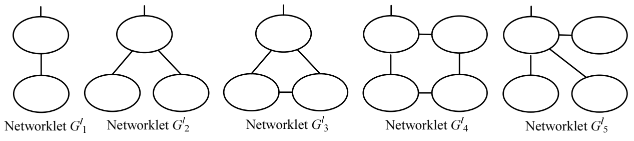


Fig. 6: Five types of networklets used in our simulations

with more complex and natural topologies of networklets. The bandwidth requirement of each edge in networklets is uniformly generated from 1 to 3. The service capacity of each networklet is uniformly generated from 15 to 20.

By default, the number of tenant requests is 1000, each of which accesses 0 to 2 networklets. The service demand for each networklet is uniformly generated from 1 to 3. The number of IaaS VMs, *i.e.*, the number of vertices in V^b, is uniformly generated from 3 to 5. Each pair of IaaS VMs communicate with a probability $p = 0.3$. We continue to generate the communication graph (*i.e.*, topology) until we get a connected one. The bandwidth requirement of each edge in HTRs is uniformly generated from 1 to 3. The lifetime of a request varies uniformly in the range from 10 to 20.

We compare the proposed algorithm, *i.e.*, HNDA, with the following two algorithms: **HoseAlg**, which converts a HTR+networklet request into a hose request and then deploys it using the allocation algorithm devised in [12], and **CliqueAlg**, which converts a HTR+networklet request into a clique request and then deploys it using Alg. 2 of this paper. Performance metrics are as follows:

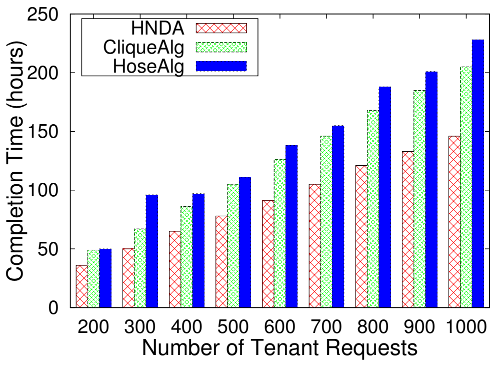
- Completion time, which is the time to complete all tenant requests. The smaller the completion time is, the better the performance is. In fact, a smaller completion time also implies that, physical resources are utilized efficiently.
- Computing resource utilization, which is defined as the ratio of the number of occupied VM slots to the total number of VM slots in the given DCN.

B. Simulation Results

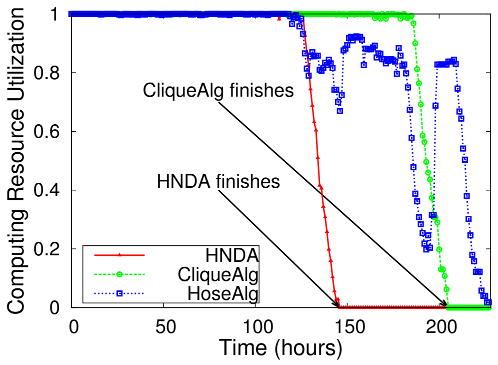
Fig. 7(a) shows the completion time achieved by three algorithms versus the number of requests. Note that, the completion time is the time to complete all tenant requests, not the running time of each deployment algorithm. In general, we find that, in all settings, the completion time of all requests using HNDA is the smallest among them, while using HoseAlg has the largest completion time. This observation is consistent with the perceptual comparison results in Table I.

When the number of requests increases from 200 to 1000, the increases in the completion time of HNDA, CliqueAlg, and HoseAlg are 110, 154, and 178, respectively, *i.e.*, the completion time using HNDA is only 64% of using HoseAlg. The increase speed of HNDA is the smallest, implying the good scalability of our model and algorithm.

Fig. 7(b) shows the computing resource utilization over time when the number of tenant requests is 1000. The completion times of these 1000 requests using HNDA, CliqueAlg, and HoseAlg are 146, 205, and 228, respectively. After HNDA or CliqueAlg finishes the batch of tenant requests, we assume their computing resource utilization is zero. We plot the



(a) Completion time comparison



(b) Computing resource utilization over time

Fig. 7: Simulation results

computing resource utilizations at the beginning of each hour. We see that, the utilization ratio of HNDA is always higher than the other two algorithms before it finishes the batch of requests; CliqueAlg achieves a better resource utilization than HoseAlg, which is especially clear from the 120th hour to the 145th hour in the figure.

We note that, the resource utilization of HoseAlg fluctuates over time. There may be some relatively large tenant request among these 1000 requests. For such a large request described in the hose model, it may demand much more bandwidth than it needs if it is described using other models. The large resource requirements make it hard to deploy in the underlying DCN, which may embarrass the underly DCN in a sense: DCN has some residual resource, but cannot afford this large request. In summary, some relatively large tenant requests make HoseAlg's computing resource utilization vary over time, which suggests the disadvantage of the hose model.

In summary, the proposed model and deployment algorithm achieve shorter completion time and better resource utilization than the hose or clique-based algorithm. We admit that the above-presented results are far from exhaustive; however, we hope these results can provide insights into modeling and deploying hybrid tenant resource requests and thus open a new avenue for resource allocation in multi-tenant datacenters.

VI. Conclusions

In this paper, we introduce networklet, a notion that helps to explore the tradeoff between performance guarantee and resource utilization in hybrid datacenters. Sharing networklets between multiple tenants achieves better multiplexing which benefits both tenants and providers. Extensive evaluations show that, extracting networklets from hybrid tenant requests indeed improves DCN resource utilization while maintaining performance guarantee.

Acknowledgments

This work was supported in part by NSFC (61502224, 61472181, 61472185, 61321491), China Postdoctoral Science Foundation (2015M570434, 2016T90444), CCF-Tencent Open Research Fund (AGR20160104), Jiangsu NSF (BK20151392, BK20151390), and Collaborative Innovation Center of Novel Software Technology and Industrialization.

References

[1] "Amazon DynamoDB," https://aws.amazon.com/dynamodb/.
[2] "Amazon ElastiCache," https://aws.amazon.com/elasticache/.
[3] "Amazon RDS," https://aws.amazon.com/rds/.
[4] "Amazon RedShift," https://aws.amazon.com/redshift/.
[5] "AWS Elastic Beanstalk," http://aws.amazon.com/elasticbeanstalk.
[6] "Engine Yard," https://www.engineyard.com.
[7] "OpenStack Trove," https://wiki.openstack.org/wiki/Trove.
[8] "QINIU," http://www.qiniu.com.
[9] "Salesforce," http://salesforce.com/platform/solutions/employee-apps.
[10] "Thoughts on Cloud," http://alturl.com/hmxbr.
[11] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica *et al.*, "A view of cloud computing," *CACM*, vol. 53, no. 4, pp. 50–58, 2010.
[12] H. Ballani, P. Costa, T. Karagiannis, and A. I. Rowstron, "Towards predictable datacenter networks," in *Proc. ACM SIGCOMM 2011*, pp. 242–253.
[13] M. Chowdhury, M. Rahman, and R. Boutaba, "ViNEYard: Virtual network embedding algorithms with coordinated node and link mapping," *IEEE/ACM TON*, vol. 20, no. 1, pp. 206–219, 2012.
[14] R. Cohen, L. Lewin-Eytan, J. Naor, and D. Raz, "Near optimal placement of virtual network functions," in *Proc. IEEE INFOCOM 2015*, pp. 1346–1354.
[15] J. Duan, Z. Guo, and Y. Yang, "Cost efficient and performance guaranteed virtual network embedding in multicast Fat-Tree DCNs," in *Proc. IEEE INFOCOM 2015*, pp. 136–144.
[16] 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 *Proc. ACM CoNEXT 2010*, pp. 1–12.
[17] J. Lee, Y. Turner, M. Lee, L. Popa, S. Banerjee, J.-M. Kang, and P. Sharma, "Application-driven bandwidth guarantees in datacenters," in *Proc. ACM SIGCOMM 2014*, pp. 467–478.
[18] Y. Liang and S. Zhang, "Embedding parallelizable virtual networks," *Computer Communications*, vol. 102, pp. 47–57, 2017.
[19] D. Xie, N. Ding, Y. C. Hu, and R. Kompella, "The only constant is change: incorporating time-varying network reservations in data centers," in *Proc. ACM SIGCOMM 2012*, pp. 199–210.
[20] M. Yu, Y. Yi, J. Rexford, and M. Chiang, "Rethinking virtual network embedding: substrate support for path splitting and migration," *ACM SIGCOMM CCR*, vol. 38, no. 2, pp. 17–29, 2008.
[21] S. Zhang, Z. Qian, J. Wu, S. Lu, and L. Epstein, "Virtual network embedding with opportunistic resource sharing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 3, pp. 816–827, 2014.
[22] S. Zhang, Y. Liang, Z. Qian, M. Xiao, J. Wu, F. Kong, and S. Lu, "Networklet: Concept and deployment," in *Proc. IEEE ICDCS 2017*, pp. 2624–2625.
[23] S. Zhang, Z. Qian, S. Guo, and S. Lu, "Fell: A flexible virtual network embedding algorithm with guaranteed load balancing," in *Proc. of IEEE ICC 2011*, pp. 1–5.
[24] S. Zhang, Z. Qian, J. Wu, and S. Lu, "Leveraging tenant flexibility in resource allocation for virtual networks," in *Proc. of IEEE ICCCN 2014*, pp. 1–8.