D-Slice: An Elastic Network Planning Scheme in Hose-Based Data Center Networks

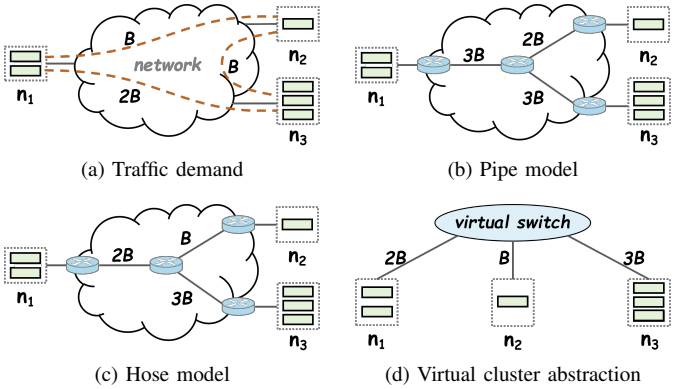
Jie Wu, *Fellow, IEEE*, Shuo Quan, *Member, IEEE*, Xiaoyao Huang, *Member, IEEE*, and Huitao Zhou, *Member, IEEE*

***Abstract*—Efficient resource management and optimization in cloud data centers require elastic scaling with bandwidth guarantees to support dynamic cloud workloads. Existing methods provide runtime guarantees through bandwidth allocation and path selection, or support growth through capacity reservation and elastic VM placement. The former lack joint compute and network reservations for growth; the latter have limitations in growth assumptions, capacity utilization, elasticity, or routing efficiency. We study elastic network planning under the hose model, aiming to place initial virtual machines (VMs) to maximize subsequent growth, termed *elasticity*, without relocating them. This problem is NP-hard even without growth. We therefore adopt a capacity-oriented provisioning strategy and propose *D-slice*, a polynomial-time maximum-flow-based framework. A D-slice is a directed acyclic graph (DAG)-shaped subgraph rooted at a switch or server and defines a hose-feasible VM-slot reservation vector together with link-bandwidth reservation. This reserved capacity provides the resource basis for elastic VM placement and guarantees hose-based traffic patterns within the slice. We further devise a flow-guided routing scheme based on the maximum-flow structure and an efficient short-cut routing scheme that reduces hop count without complex routing tables while preserving bandwidth feasibility. We also study the composition of multiple D-slices in multi-root spine-leaf DCNs, where compatible slices can be merged to increase supportable capacity, and extend the framework to a two-level spine-leaf metropolitan area network. Extensive simulation and testbed evaluations show the efficacy of D-slice and its flow-guided routing in hose-based DCNs.**

***Index Terms*—Data center network, resource management, joint cloud-network planning, resource provisioning, hose model, routing, virtual machine.**

I. Introduction

In cloud computing, resource management and optimization are fundamental to improving resource utilization and service performance in multi-tenant environments. However, cloud providers [1] often offer tenant network resources on a best-effort basis and cannot provide explicit guarantees on the network bandwidth available to each virtual machine (VM) hosted on a server. Under the vision of *cloud-network convergence*, joint cloud-network planning [2] addresses this issue by coordinating cloud resource allocation and network provisioning. A key challenge in bandwidth-guaranteed network planning [3] is handling unpredictable tenant traffic. Compared with the *pipe* model [4], which requires a pairwise traffic matrix, the *hose* model [5] specifies only the aggregate ingress/egress bandwidth of each server hosting multiple VMs. In this setting, VMs act as CPU-memory integrated scheduling units. This abstraction simplifies bandwidth specification and improves flexibility, but it also makes network planning more challenging at large deployment scales.



(a) Traffic demand (b) Pipe model (c) Hose model (d) Virtual cluster abstraction

Fig. 1. Pipe vs. hose bandwidth provisioning in DCNs.

Fig. 1 illustrates a simple data center network (DCN) where servers (rectangles) are interconnected through switches (circles). Assuming that each VM has a bandwidth budget of B Mbps, Fig. 1(a) shows one possible communication demand. Under the pipe model in Fig. 1(b), bandwidth is reserved for each communicating VM pair. For example, the outgoing link of n_1 must support B traffic to n_2 and $2B$ traffic to n_3, requiring a total reservation of $3B$. In contrast, under the hose model in Fig. 1(c), since two VMs are hosted on n_1, the aggregate ingress/egress traffic of n_1 is bounded by $2B$, and only $2B$ needs to be reserved. Fig. 1(d) further shows the virtual-cluster abstraction, where n_1, n_2, and n_3 connect to a virtual switch with bandwidths $2B$, B, and $3B$, respectively. It supports any pairwise communication pattern that satisfies these aggregate bandwidth limits. For example, it can support either the pairs (n_1, n_2), (n_1, n_3), and one local pair within n_3, or two (n_1, n_3) pairs and one (n_2, n_3) pair, without changing the reserved bandwidth. Therefore, compared with the pipe model, the hose model better accommodates dynamic communication patterns, avoids the use of complex traffic matrices, and enables statistical multiplexing, thereby improving bandwidth utilization under dynamic tenant workloads.

Although the hose model provides bandwidth guarantees, satisfying only current demand is insufficient in practical DCNs, where modern cloud services increasingly require elastic scaling [6]–[8]. As applications scale and communication demands continue to grow, especially in emerging AI and

Jie Wu, Shuo Quan, Xiaoyao Huang, and Huitao Zhou are with the Cloud Computing Research Institute, China Telecom, Beijing, China (e-mail: jiewu, Quansh, huangxy32@chinatelecom.cn, zhouhuitao@bupt.edu.cn.)

Jie Wu is also with the Center for Networked Computing, Temple University, USA (e-mail: jiewu@temple.edu).

cloud workloads [9]–[11], network planning should reserve capacity for future expansion. Therefore, we study the following problem: under hose-model bandwidth guarantees, how should current VMs be placed on servers so that the deployment can accommodate future VM and communication growth without re-scheduling? This maximum-elasticity scheduling problem is challenging. Even without growth, determining whether a given number of VMs can be placed within server capacities with bandwidth reservations that support every communication pattern satisfying the hose bandwidth budgets is NP-hard in general topologies. Elastic growth further complicates the problem by requiring coordination between initial placement and resource reservations to avoid limiting subsequent expansion. We therefore adopt a two-step provisioning strategy: first construct a hose-feasible reservation with as much VM-slot capacity as possible and corresponding bandwidth guarantees; then place the current VMs within the reserved slots to maximize elasticity. The first step reserves capacity for growth, while the second places VMs according to their growth requirements to make effective use of that capacity.

To address this challenge, we propose *D-slice*, a maximum-flow-based network slicing method that jointly organizes VM-slot reservations on servers and bandwidth reservations in the switch network. A D-slice is a DAG-shaped resource allocation structure rooted at a switch or server. It is constructed by computing the maximum flow from the root to a super-sink connected to all servers, yielding a VM-slot reservation vector and the corresponding bandwidth reservation. Unlike traditional network slicing approaches that mainly partition network resources into isolated logical slices, D-slice jointly captures VM-slot reservation, bandwidth reservation, elastic placement, and routing structure within the DCN.

Under certain conditions, multiple single-root D-slices can be composed, either directly or after trimming, into a larger multi-root compound slice, called an MD-slice (Merged D-slice). An MD-slice further improves resource utilization and supportable VM-slot capacity, making it suitable for large-scale hose-based applications. Both D-slices and MD-slices support a unified two-phase routing strategy: up toward the root and down toward the destination, following the direction and capacity induced by the maximum flow.

Finally, the D-slice framework is extended to multi-level networks, such as a metropolitan area network (MAN) with a structured two-level spine-leaf topology. Extensive simulations and testbed experiments demonstrate the efficacy of D-slice, while comparisons with traditional shortest-path routing validate the effectiveness of the corresponding flow-guided short-cut routing in DCNs. The proposed D-slice applies to general network topologies and is not limited to spine-leaf DCNs. The main contributions can be summarized as follows.

- We propose *D-slice*, a maximum-flow-based network-planning scheme that jointly reserves server-side VM slots and link bandwidth, providing a hose-feasible resource basis for elastic VM placement.
- We design a simple flow-guided routing scheme and an enhanced short-cut routing scheme, both preserving D-slice bandwidth feasibility while reducing routing complexity and hop count.

TABLE I
THE SUMMARY OF MAIN NOTATIONS

$G = (V, E)$	a DCN graph
N, S	the sets of server and switch nodes in G
M, u, r	the requested VM set, a VM, and initial VM count
B	the basic bandwidth unit of each VM
e	the elasticity of a VM deployment
g_u	the growth rate of VM u
A_i	the set of initial VMs placed on server n_i
v_i	the VM-slot capacity reserved on server n_i
r_i	the number of initial VMs placed on server n_i
$d_i(e)$	the demand of server n_i under elasticity e
$c(l)$	the capacity of link l in units of B
$c(n_i)$	the VM capacity of server n_i
cut_k	a cut partitioning V into two disjoint sets
$D(\mathrm{src})$	the root of a D-slice
$D(\mathrm{ser})$	the server set of a D-slice
α	intra-DCN traffic ratio in the mixed hose model

- We introduce *MD-slice* for multi-root DCNs, establish a sufficient condition for safe D-slice composition, and design the corresponding routing method.
- We extend D-slice to multi-level networks using MAN as an example, where a two-level spine-leaf structure supports heterogeneous traffic.
- We conduct extensive simulations and testbed experiments on structured and unstructured topologies to demonstrate the efficacy of the proposed scheme.

II. PRELIMINARIES

This section presents the graph and elasticity models for DCNs and formulates a network planning and scheduling problem termed maximum-elasticity scheduling.

A. Graph Model for Networks

A DCN is represented as a graph $G = (V, E)$, where $V = N \cup S$ denotes the node set, comprising the server set N and the switch set S, and E denotes the set of physical links in the Table I. Each link $l \in E$ has a bandwidth capacity $c(l)$, expressed as a multiple of the base bandwidth B Mbps. Each server node n has capacity $c(n)$, representing the maximum number of VMs it can host. Each VM is assigned a bandwidth budget of B. Under the hose model, each VM may communicate with any other VMs, provided that its aggregate outgoing and incoming traffic rates are each bounded by B. The communicating pairs and their individual traffic rates are not specified in advance and may vary over time. A feasible resource reservation must support every traffic matrix satisfying these per-VM bounds. Thus, a server hosting k VMs has aggregate outgoing and incoming traffic each bounded by kB. This setting is consistent with practical cloud instance specifications[1]. As shown in Fig. 1, rectangles denote servers, circles denote switches, and straight lines denote communication links.

[1]Same as our model here, AWS EC2 instances provide approximately linearly increasing network bandwidth with the number of vCPUs [12].

B. Elasticity Model

Let M denote the set of VMs in the current request. We use $e \geq 0$ to denote the elasticity of a deployment. Since different VMs may have different scaling speeds, each VM $u \in M$ is associated with a growth rate $g_u \geq 0$, which represents the additional number of VMs required by the service corresponding to u when the elasticity level increases by 1. Therefore, under elasticity e, the overall demand of VM u is defined as $1 + eg_u$. Here, the constant 1 represents the current base demand, and eg_u represents the additional demand caused by future scaling. A VM allocation is represented as $A = \{A_i\}$, where A_i is the set of VMs assigned to server n_i. By definition, A is a partition of M, meaning that each VM is assigned to exactly one server. For server n_i, define $G_i = \sum_{u \in A_i} g_u$. The total demand of server n_i under elasticity e is $d_i(e) = \sum_{u \in A_i}(1 + eg_u) = |A_i| + eG_i$.

C. Maximum Elastic Scheduling

This paper studies the maximum-elasticity scheduling problem under the hose model. Given a DCN G, a VM set M, and growth rates g_u, the objective is to find a VM deployment that can support the maximum elasticity while satisfying server capacity and link bandwidth constraints. Let $\mathbf{b} = \{b(l)\}$ denote the link bandwidth reservation scheme, where $0 \leq b(l) \leq c(l)$ for each link l. For any cut k in the network, let cut_k denote the set of links crossing this cut, and let $\text{cut}_{k.l}$ and $\text{cut}_{k.r}$ denote the sets of server nodes on the two sides of this cut, respectively. The maximum-elasticity scheduling problem can be formulated as

$$\underset{A,\mathbf{b}}{\text{maximize}}\ e \tag{1}$$

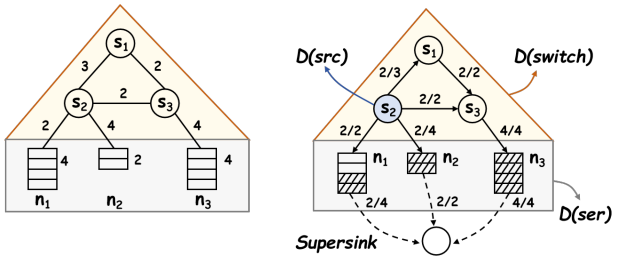
$$s.t. \quad d_i(e) \leq c(n_i) \text{ for } \forall n_i \in N, \tag{2}$$

$$0 \leq b(l) \leq c(l) \text{ for } \forall l \in E, \tag{3}$$

$$\sum_{l \in \text{cut}_k} b(l) \geq \min\{\sum_{n_i \in \text{cut}_{k.l}} d_i(e), \sum_{n_i \in \text{cut}_{k.r}} d_i(e)\} \text{ for } \forall k. \tag{4}$$

Constraint (2) requires that the total demand of each server under elasticity e does not exceed its server capacity. Constraint (3) ensures that the reserved bandwidth on each link is within its physical capacity. Constraint (4) is the hose cut constraint: for any cut k, the reserved bandwidth across the cut must be no smaller than the worst-case hose traffic crossing it. This worst-case traffic equals the smaller aggregate demand of the two sides: $\min\left\{\sum_{n_i \in \text{cut}_{k.l}} d_i(e), \sum_{n_i \in \text{cut}_{k.r}} d_i(e)\right\}$. Thus, the constraint bounds the traffic of all VM pairs separated by the cut without requiring a pairwise traffic matrix.

The above cut-based formulation contains an exponential number of cut constraints, making it difficult to solve directly in general topologies. To characterize its hardness, we consider the related hose-based capacity-planning problem, which asks for the maximum number of VM slots supportable under server capacities and hose bandwidth constraints. This problem is NP-hard in general topologies and establishes the hardness of maximum-elasticity scheduling: when $e = 0$, the latter reduces to deciding whether the current VM set can be placed under the same server and hose bandwidth constraints. The formal statements and proofs are presented in Section III-B.



(a) DCN topology. (b) D-slice rooted at s_2.

Fig. 2. An example of constructing a D-slice with server set $\{n_1, n_2, n_3\}$ from a DCN topology using root s_2.

However, when the graph G is a tree as in a typical DCN [13], each link l induces exactly one cut cut_k. Therefore, the number of cut constraints becomes linear in the number of links, i.e., $|E| = |V| - 1$. In this case, the objective and constraints remain the same as in (1)-(4), except that each cut_k in (4) is the cut induced by removing link l_k from the tree. Thus, the cut-bandwidth constraint can be written as

$$b(l_k) \geq \min\{\sum_{n_i \in \text{cut}_{k.l}} d_i(e), \sum_{n_i \in \text{cut}_{k.r}} d_i(e)\} \text{ for } \forall\ l_k. \tag{5}$$

Although this tree-based formulation contains only $O(|V|)$ constraints, it can still be further simplified. In fact, there exists a simple algorithm with complexity $O(\Delta|V|)$, where Δ is the maximum degree of the tree. When the given tree is binary, the complexity becomes linear in $|V|$ [13].

III. Models and Solutions

In this section, we introduce the D-slice as a network planning tool for maximum-elasticity scheduling, analyze the problem complexity, present its max-flow-based construction for general DCNs, and develop elastic VM placement methods.

A. D-slice Model

We propose a new network slicing method specifically designed for the hose model that involves two entities: VMs and their communication bandwidth requirement under the hose model. This slicing method offers network planning for both virtual machines (typically done in the cloud) and network link scheduling that guarantees bandwidth. In addition, the slice construction offers a convenient way of routing in any pairwise VM communication.

Given a network G, a D-slice is a DAG-shaped subgraph of G that reserves both server capacity and link bandwidth. As shown in Fig. 2(b), a D-slice consists of a source switch/server as the root, a set of servers at the bottom layer, and directed links connecting them. The root is introduced for both construction and routing. For a slice D, $D(src)$ denotes its root, $D(ser)$ denotes its server set, and $D(switch)$ denotes its switch component. For each server $n_i \in D(ser)$, let v_i denote the VM-slot capacity reserved on n_i. The vector $(v_1, \ldots, v_k)$ represents the VM-slot reservation vector over $D(ser)$, and the resulting D-slice satisfies both server and bandwidth constraints under the hose model.

For example, suppose six VM slots are reserved on the servers in Fig. 2(a). The reservation vector (n_1, n_2, n_3) :

$(2, 2, 2)$ satisfies the server constraints, and there exists a feasible bandwidth reservation that supports arbitrary VM communication among the three servers without exceeding link capacities. Different choices of $D(src)$ may produce different D-slices for the same physical network.

Note that D-slice is defined over a general graph, with a tree topology being only a special case. Popular structured DCNs, such as Fat-tree and Spine–Leaf, can also be represented as logical trees through equal-cost multipath (ECMP)-based path selection and therefore naturally fit into the D-slice framework.

B. Problem Complexity

Before presenting the construction and placement methods, we establish the complexity of hose-based capacity planning and maximum-elasticity scheduling.

***Theorem** 1:* In general topologies, computing the maximum number of VMs that can be supported under the hose model, together with a feasible bandwidth reservation, is NP-hard.

Proof: Consider the following hose-based capacity feasibility problem. Given a general network topology, link bandwidth constraints, and a hose demand vector $\mathbf{h} = \{h_i\}$ on the servers, determine whether there exists a routing and bandwidth reservation scheme that can support all traffic matrices satisfying the hose-model constraints. This problem is equivalent to the capacitated hose routing/virtual private network (VPN) provisioning feasibility problem in general graph topologies, which has been proven to be NP-hard [14], [15].

We reduce this feasibility problem to the hose-based capacity-planning problem considered in Theorem 1. Given an instance with hose demand vector $\mathbf{h}$, construct a capacity-planning instance by setting the server capacity of each server n_i to $c(n_i) = h_i$, while keeping the same network topology and link capacities. Let $C^\star$ be the maximum supportable capacity returned by an algorithm for the capacity-planning problem. Since each reserved VM-slot capacity v_i must satisfy $v_i \le c(n_i) = h_i$, the original hose feasibility instance is feasible if and only if $C^\star = \sum_i h_i$. Therefore, a polynomial-time algorithm for computing the maximum supportable capacity would solve the NP-hard hose feasibility problem in polynomial time. Hence, the hose-based capacity-planning problem is NP-hard. ■

***Corollary** 1:* In general topologies, the maximum-elasticity scheduling problem under the hose model is NP-hard.

Proof: We prove NP-hardness by considering the case $e = 0$. Given an instance of the hose-based capacity-planning decision problem with target capacity r, construct a maximum-elasticity scheduling instance containing r current VMs with arbitrary nonnegative growth rates. When $e = 0$, the demand on server n_i becomes $d_i(0) = |A_i|$, so all growth terms vanish. Hence, supporting these r VMs with elasticity at least zero is equivalent to supporting at least r VM slots under the same server and hose-bandwidth constraints. Since the former would solve the NP-hard capacity-planning problem of Theorem 1, maximum-elasticity scheduling is NP-hard. ■

Algorithm 1 D-slice Construction

Input: A graph G with a selected source
Output: Maximum VM-slot capacity D-slice over $D(ser)$

1: Convert the graph into a flow network with a supersink connected to all servers.
2: Set each server-to-supersink link capacity to the corresponding server capacity.
3: Find the **maximum flow** from the source to the supersink.
4: Remove directed flow cycles by repeatedly subtracting the minimum flow on each detected cycle from all its edges.
5: Use the flow from each server to the supersink as the VM-slot capacity reserved on that server.

C. D-slice Construction

A D-slice can be constructed based on the classic maximum-flow formulation. Given a selected root, the original graph is converted into a flow network by introducing a special node, called the *supersink*, which is connected to each server. The capacity of each server-to-supersink link is set to the corresponding server capacity, as shown in Fig. 2(b). Algorithm 1 details the construction for a selected root. It computes the maximum flow from the root to the supersink and removes any directed cycles formed by positive-flow edges before determining the reservations. For each detected cycle, the minimum flow on the cycle is subtracted from every edge on it, repeating until no directed cycle remains. This preserves flow conservation and the maximum-flow value. Since the supersink has no outgoing edges, server-to-supersink flows remain unchanged and are used as the reserved VM-slot capacities. After removing the supersink and its incident edges, the remaining positive-flow subgraph is a DAG, as required by the D-slice definition.

Note that the maximum-flow based VM-slot reservation at each server does not necessarily depend solely on the server capacity. For example, with s_2 selected as the root in Fig. 2(b), the maximum flow is 8 units, yielding the VM-slot reservation vector $(n_1, n_2, n_3) : (2, 2, 4)$. Server n_1 can reserve only 2 VM slots although it has a capacity of 4, because its connection to s_2 has a capacity of only 2; reserving more VMs would require more bandwidth on this link under the hose model, otherwise arbitrary VM communication cannot be guaranteed. Similarly, n_2 is limited to 2 VM slots by its server capacity, while the maximum-flow allocation assigns 4 VM slots to n_3.

The following theorem states two key properties of D-slice construction: root-wise maximum VM-slot capacity and hose-feasible communication.

***Theorem** 2:* The D-slice construction builds a D-slice with the maximum cardinality (i.e., VMs' number) for a given root via the maximum flow. In addition, the link capacity allocations support any type of pairwise VM communication.

Proof: By the D-slice construction, the original graph is converted into a flow network with root D(src) as the source and a supersink connected to all servers in D(ser). The capacity of each server-to-supersink edge is the server capacity.

Therefore, each unit of flow represents one reserved VM slot, and the flow value from server n_i to the supersink gives the reserved VM-slot capacity v_i on that server.

We first show root-wise optimality for a fixed root. Any feasible D-slice rooted at D(src) with VM-slot reservation vector $(v_1, \ldots, v_k)$ corresponds to a feasible flow of value $\sum_i v_i$ from the root to the supersink, because the reserved slots must be supported by both server capacities and link capacities. If Algorithm 1 did not produce the maximum VM-slot capacity for this root, then there would exist another feasible D-slice with a larger total reservation, which would imply a feasible flow larger than the computed maximum flow. This contradicts the optimality of maximum flow.

We next show simultaneous routing feasibility for any hose-compliant traffic matrix. Decompose the maximum flow into root-to-server paths with associated bandwidth reservations, totaling $v_i B$ for each server n_i. Each link's reserved bandwidth is available independently in both directions, so these paths support both server-to-root and root-to-server communication.

For each demand, connect a source-side upstream path to a destination-side downstream path through the root. Allocate the minimum of the remaining demand and the residual bandwidth on the two paths, deducting it from the source-side upstream reservation and the destination-side downstream reservation. If demand remains, continue with available path pairs, allowing the demand to be split across multiple paths. The hose constraints separately bound each server's total outgoing and incoming demands by $v_i B$, ensuring sufficient reservations for all demands. On every link, the sum of the path reservations traversing it does not exceed its reserved bandwidth. Upstream and downstream traffic are bounded separately by this sum in their respective directions. Therefore, simultaneous communication among multiple pairs cannot overload any link. ■

To obtain the maximum single-root D-slice, each node is considered as a candidate root, and a maximum-flow computation is performed for each candidate. The computational complexity therefore depends on the adopted maximum-flow algorithm. For example, the classical Edmonds–Karp algorithm has complexity $O(|V||E|^2)$ [16], while recent advances achieve almost-linear running time $|E|^{1+o(1)}$ for polynomially bounded integral capacities [17][2]. Since maximum-flow algorithms are well studied, we focus on the resulting D-slice rather than the specific flow algorithm.

The proposed D-slice construction extends the tree-based solution in [13] from tree topologies to general graphs. For a tree, the unique-path structure enables an exact solution in $O(\Delta|V|)$ time, where Δ is the maximum node degree, and the resulting D-slice achieves the maximum capacity.

D. Elastic VM Placement

After constructing a D-slice, requested VMs are placed within its reserved slots. When $g_u = 0$ for all VMs, slots and bandwidth are allocated according to the current request size.

[2]Here, $o(1)$ denotes a term tending to zero as $|E|$ grows, so $|E|^{1+o(1)}$ is linear up to a subpolynomial factor.

Algorithm 2 Uniform Growth Elastic VM Placement

Input: D-slice D, requested VM number r
Output: Current VM counts $\{r_i\}$

```
Initialize r_i ← 0 for all servers n_i ∈ D(ser)
for t = 1 to r do
    Select i = arg max_j (v_j/(r_j+1) − 1) where r_j + 1 ≤ v_j
    r_i ← r_i + 1
end for
```

The corresponding resource allocation and reclamation methods are discussed in [18]. When future growth is required, VM placement determines the achievable elasticity. We consider two growth patterns: uniform and nonuniform growth.

1) Uniform growth: When each VM exhibits uniform growth, the VM load on each server n_i can scale up from r_i to the reserved capacity v_i — the value derived from the max-flow computation at server n_i — where r_i denotes the number of VMs placed on server n_i and r denotes the total number of requested VMs. To maximize the overall elasticity, the ratio v_i/r_i should be balanced across servers, subject to $\sum_i r_i = r$. To resolve fractional-assignment artifacts, we adopt a greedy incremental augmentation strategy.

The resulting greedy procedure is given in Algorithm 2. Starting with $r_i = 0$ for all servers, each iteration assigns one VM to a feasible server. If the next VM is assigned to server n_i, its VM count becomes r_i+1; substituting r_i+1 for r_i in the above elasticity bound gives the resulting elasticity $v_i/(r_i + 1) - 1$. The algorithm therefore selects the feasible server with the largest resulting elasticity, subject to $r_i + 1 \leq v_i$, and continues until all r VMs are placed.

For example, consider the D-slice in Fig. 2(b) with reservation vector $(n_1, n_2, n_3) : (2, 2, 4)$ and four requested VMs. Initially, the resulting elasticities for assigning the first VM to n_1, n_2, and n_3 are $(1, 1, 3)$, respectively, so the first VM is assigned to n_3. The corresponding values then become $(1, 1, 1)$; breaking the tie by assigning the next VM to n_1, they become $(0, 1, 1)$. Assigning the third VM to n_2 leaves $(0, 0, 1)$, and the fourth VM is therefore assigned to n_3. The resulting placement is $(1, 1, 2)$, yielding elasticity $e = \min\{2/1-1, 2/1-1, 4/2-1\} = 1$, meaning that the workload can double without re-scheduling. In contrast, placements such as $(2, 1, 1)$ and $(1, 2, 1)$ have zero elasticity because one of the first two servers is already fully occupied. The following theorem shows that this greedy placement achieves the maximum elasticity under uniform growth.

***Theorem** 3:* Under uniform future growth, the VM placement method in Algorithm 2 achieves maximum elasticity.

Proof: For each server n_i, consider the finite ordered sequence $(v_i - 1)/1, (v_i - 2)/2, \ldots, (v_i - v_i)/v_i$, where the j-th value represents the resulting elasticity after placing the j-th VM on server n_i. Equivalently, this value can be written as $v_i/j - 1$. Placing r VMs is equivalent to selecting a prefix of length r_i from each server sequence, with $\sum_i r_i = r$. Since each sequence is non-increasing, the elasticity of the final deployment is determined by the minimum selected value. To maximize

Algorithm 3 Nonuniform Growth Elastic VM Placement

Input: D-slice D, VM set M with growth rates $\{g_u\}_{u\in M}$
Output: Current VM assignment $\{A_i\}$

Initialize $A_i \leftarrow \emptyset$, $r_i \leftarrow 0$, and $G_i \leftarrow 0$ for all $n_i \in D(\mathrm{ser})$
Sort VMs in non-increasing order of g_u
for each VM $u \in M$ **do**
  Select $i = \arg\max_{j:\, r_j+1\le v_j} \frac{v_j - r_j - 1}{G_j + g_u}$
  $A_i \leftarrow A_i \cup \{u\}$
  $r_i \leftarrow r_i + 1$
  $G_i \leftarrow G_i + g_u$
end for

this minimum value, one should always select the currently largest available next value among all servers.

Algorithm 2 follows exactly this rule. When server n_i currently hosts r_i VMs, placing one more VM gives the score $v_i/(r_i+1) - 1$. This score has the same ordering as $v_i/(r_i+1)$, and the constraint $r_i + 1 \le v_i$ ensures that only feasible placements are considered. Therefore, no other feasible placement can achieve a larger bottleneck elasticity, which is determined by the minimum server elasticity, and the algorithm is optimal under uniform future growth. ■

2) Nonuniform growth: For VMs with nonuniform growth, the problem becomes substantially more intricate owing to heterogeneous VM growth rates. Indeed, this problem is NP-hard. Nevertheless, a simple greedy algorithm with a bounded approximation ratio is provided.

Let A_i be the set of VMs placed on server n_i, let $r_i = |A_i|$, and define $G_i = \sum_{u\in A_i} g_u$ as their aggregate growth rate. If $G_i > 0$, the elasticity supported by server n_i is $(v_i - r_i)/G_i$; if $G_i = 0$, this server does not constrain future growth. Therefore, the deployment elasticity is $e = \min_i (v_i - r_i)/G_i$.

Because both the VM counts and their growth rates must be balanced across servers, the optimality of the uniform growth method no longer holds. We therefore adopt the growth-aware greedy method in Algorithm 3. VMs are processed in non-increasing order of their growth rates, and each VM is assigned to the server that maximizes the resulting elasticity among those satisfying $r_j + 1 \le v_j$, ensuring sufficient capacity for its placement. For these feasible candidates, the score is set to $+\infty$ when $G_j + g_u = 0$. The slot-capacity check also applies to zero-growth VMs, since each still occupies one VM slot even though it requires no additional capacity for future growth. When all $g_u = 1$, we have $G_j = r_j$, and Algorithm 3 reduces to Algorithm 2.

We next analyze its performance under heterogeneous growth rates. Let $C = \sum_i v_i$ denote the total D-slice capacity, and define $g_{\max} = \max_{u\in M} g_u, g_{\min} = \min_{u\in M} g_u, \lambda = g_{\max}/g_{\min} \ge 1$. Using the conservative upper bound $1 + e g_{\max}$ for the demand of each VM, a sufficient condition for accommodating all VMs is $C \ge \big(r + |D(\mathrm{ser})|\big)(1 + e g_{\max})$, where the additional $|D(\mathrm{ser})|$ term accounts for worst-case placement fragmentation. Hence, the following elasticity is conservatively guaranteed: $e' = (C/(r + |D(\mathrm{ser})|) - 1)/g_{\max}$.

In an ideal continuous placement without fragmentation, the optimal elasticity $e^\star$ satisfies $e^\star \le (C - r)/\sum_{u\in M} g_u$. Since $\sum_{u\in M} g_u \ge r g_{\min} = (r g_{\max})/\lambda$, we obtain $e^\star \le (\lambda(C - r))/(r g_{\max})$. Combining the above bounds yields

$$\frac{e'}{e^\star} \ge \frac{r}{\lambda(C-r)} \left(\frac{C}{r + |D(\mathrm{ser})|} - 1 \right).$$

The bound decreases as λ increases, showing that stronger growth heterogeneity weakens the elasticity guarantee under worst-case provisioning.

Algorithm 4 Short-Cut Routing in a D-slice

Input: D-slice with a root, routing source s, destination d.
Output: A short-cut route through an internal switch, with flow information updated at each switch along the path.

(**Up phase**) From source s through a valid upward link until reaching a switch with a valid downward link to destination d.
(**Down phase**) From the switch through valid downward links until reaching d.
(**Switch update**) Update states of all switches along the up-phase path, the down-phase path, an up-path from the switch to the root, and a down-path from the root to the switch as an atomic action.

IV. Routing Scheme

This section presents simple and short-cut routing schemes over a D-slice, proves their bandwidth feasibility, and describes their SDN implementation and runtime updates.

A. Simple and Short-Cut Routing

D-slice offers a simple way of routing by means of a maximum flow construction. In addition, we assume that each switch has the software-defined networking (SDN) capability to record information. All links have two orientations with respect to the root (src) of the D-slice: *up* (closer to the root) and *down* (away from the root). With this orientation, each switch maintains information about its adjacent links and its reachable destinations. For example, in Fig. 3 switch s_1 keeps 5 for the upward link to s_3 and 5 for the downward link to s_4 with a reachable destination set of $\{n_3, n_4\}$. Here, we consider two routing schemes: *simple routing* and *short-cut routing*.

In simple routing, the algorithm follows an up-phase through any valid upstream link (i.e., with capacity) to the root followed by a down-phase through any legitimate downstream link (i.e., with capacity and that can reach the destination) to the destination server. Again, the initial capacity is determined through the D-slice construction using the maximum flow. Once routing is complete, update the states of all switches along the up-phase path and down-phase path to avoid simultaneous routing updates. Likewise, the bandwidth is released after the communication completes. An atomic action [19] ensures that changes in switches are completely applied or completely undone, preventing partial or inconsistent states. This atomic action can also be done through a reservation process before routing. Fig. 3 shows a simple routing between the VMs in n_3 and n_4, where the routing paths are colored blue. s_3 is the root of the D-slice.

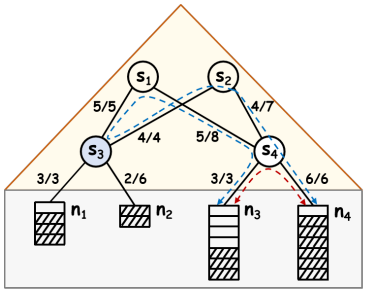


Fig. 3. Simple (blue) vs. short-cut (red) routing.

Algorithm 4 presents the short-cut routing procedure. During the up phase, a packet is forwarded toward the root until it reaches an intermediate switch that has a valid downward path to the destination. The packet then enters the down phase and is forwarded to the destination server. Note that the up-path and down-path between this intermediate switch and the root may not follow the same physical links. Fig. 3 illustrates an example between VMs on n_3 and n_4, where s_4 serves as the phase-change switch. In this case, short-cut routing reduces the path length from 6 hops to 2 hops. To maintain consistent capacity states, the controller also updates the path from s_4 to the root and the path from the root back to s_4, in addition to the two server-facing links connecting s_4 to n_3 and n_4. Each involved link decreases its residual capacity by one unit. By avoiding unnecessary traversal through the root, short-cut routing reduces bandwidth consumption, preserves more residual bandwidth under over-provisioning, and better accommodates bursty traffic, as shown in Section VII.

The following theorem establishes the bandwidth feasibility of both routing schemes under the D-slice reservation.

***Theorem** 4:* For any hose-compliant communication pattern, neither simple routing nor short-cut routing exceeds the bandwidth reserved by the corresponding D-slice.

Proof: The D-slice construction reserves link capacities according to the maximum-flow solution. In simple routing, each communication request is routed through an up-phase from the source server toward the root and a down-phase from the root toward the destination server. Since the available upstream and downstream capacities are initialized from the reserved D-slice capacities and are decreased after each routed communication, a request is admitted only when sufficient residual capacity exists on all involved links. Therefore, simple routing never exceeds the bandwidth reserved by the D-slice.

For short-cut routing, the actual forwarding path may switch from the up-phase to the down-phase at an intermediate switch before reaching the root. This only shortens the physical forwarding path. To keep the residual-capacity state consistent with the original root-oriented reservation, the controller also updates the corresponding up-path from the intermediate switch to the root and the down-path from the root back to the intermediate switch as an atomic action. Thus, the capacity accounting of short-cut routing is no larger than that of the corresponding simple route through the root. Since simple routing is feasible under the D-slice reservation, short-cut routing also cannot exceed the reserved bandwidth.

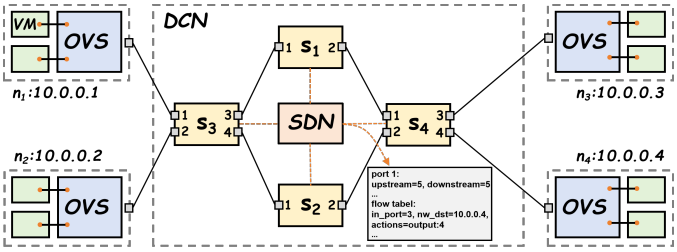


Fig. 4. Operations in the SDN switches.

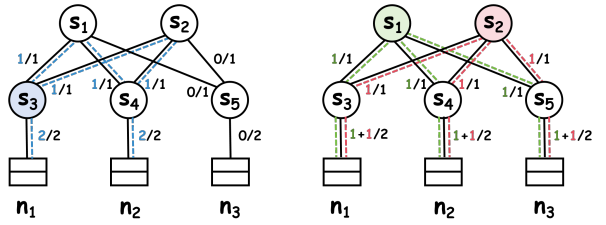
Therefore, for any hose-compliant communication pattern, both simple routing and short-cut routing respect the bandwidth reserved by the corresponding D-slice. ■

B. Operations in the SDN Switches

1) The Initial Configuration for the Switches: The SDN controller can deliver rules from the flow table to the switches according to the topology determined by the maximum flow with bandwidth capacity. First, two types of routing should be configured for the switches. For upstream switches, if there exists a link from switch s to switch s', add a route to the root in switch s. For switch s', add the server node IP address information connected to switch s. Fig. 4 shows this process for the flow example in Fig. 3, the upstream link from n_4 to the root s_3 is (n_4, s_4, s_1, s_3) and (n_4, s_4, s_2, s_3), then the rules sent to the switches by the SDN. Furthermore, the upstream and downstream capacities are sent to the corresponding switches to store. For the same example, switch s_4 configures the downstream capacity of the output ports n_4 to 6 and the upstream capacity of the output port s_1 and s_2 to 5 and 4, respectively. In reverse, the downstream capacity for s_1 and s_2 is set to 5 and 4, respectively.

2) The Runtime Routing and Updating Process for the Switches: The switch port capacity is updated according to the traffic forwarding phase. In the upstream phase, forwarding traffic decreases the upstream capacity of the output port and the downstream capacity of the input port. Conversely, in the downstream phase, the downstream capacity of the output port and the upstream capacity of the input port are reduced after forwarding. For a communicating VM pair, traffic is first processed in the upstream phase. If a valid route exists and the available downstream capacity of the selected output port is sufficient, the traffic is forwarded and the corresponding port capacities along the path are updated. Otherwise, the switch selects a feasible port with sufficient upstream capacity toward the destination and continues forwarding. This process repeats until the traffic reaches the downstream phase, where it is routed to the destination server and the remaining port capacities are updated accordingly.

The above operations are implemented using match-action tables and registers in programmable switches: match-action tables determine forwarding ports based on destination addresses, while registers maintain residual upstream and downstream port capacities, with the corresponding configuration managed by the control plane [20], [21]. Within a switch, capacity checks and updates use supported atomic read-modify-write operations [22]. Reservation updates across switches are



(a) Single-root D-slice. (b) Composed D-slice.

Fig. 5. A motivating example of slice composition. (a) Using s_3 as the root, the max-flow-based method obtains a D-slice of capacity 4. (b) Two D-slices generated from roots s_1 and s_2 are composed into a compound slice of capacity 6.

coordinated by the controller, with consistent-update mechanisms supporting forwarding-rule transitions [21], [23]. Building on these mechanisms, D-slice maintains direction-specific capacity states so that short-cut routing shortens the actual forwarding path while retaining capacity accounting along the corresponding root-based route.

Maintaining these capacity states introduces overhead from capacity checks, state updates, and coordination across switches. On programmable switches, measured median table-update latencies are approximately $22\,\mu$s with a vendor-specific C++ interface and 262–590 μs with P4Runtime implementations [20]. These measurements provide a reference for the rule-update cost. In D-slice, control-plane updates establish or adjust communication reservations, which subsequent messages reuse. Switch-local capacity checks and updates do not require a separate control-plane commit for each packet. The setup cost is therefore amortized over the communication using the reservation, making the overhead acceptable under this reservation-reuse scheme.

V. Slice Composition

This section studies how to build a larger compound slice from multiple D-slices, derives the corresponding composability condition, and describes routing after composition.

A. Composability Condition

In the previous section, a D-slice is generated in a general graph by a max-flow-based method with a single root. In multi-root topologies, such as Spine–Leaf and Fat-tree DCNs, different roots may produce different feasible D-slices. We next study whether these D-slices can be composed into a larger compound slice while still satisfying the hose bandwidth guarantee. Fig. 5 gives a motivating example. In Fig. 5(a), using s_3 as the root, the max-flow-based method obtains a D-slice with capacity 4. In contrast, Fig. 5(b) shows that two D-slices generated from roots s_1 and s_2 can be composed into a compound slice with capacity 6. Thus, slice composition can exploit multiple roots to obtain a larger reservation than any single-root solution.

Under the hose model, if a cut k separates $cut_{k.l}$ reserved VM slots on one side and $cut_{k.r}$ reserved VM slots on the other side, the maximum possible traffic across this cut is $B{\cdot}\min\{cut_{k.l}, cut_{k.r}\}$. Therefore, slice composition is feasible only if, after composition, the maximum traffic crossing any cut is no larger than the total bandwidth provided by the involved slices on this cut.

For a general graph, directly checking all cuts can be expensive. In a tree, however, each edge induces a simple cut: after removing an edge, all traffic between the two resulting sides must traverse this edge. Therefore, if multi-root slice composition can be transformed into tree-based checks, feasibility can be verified through edge-by-edge cut constraints. As shown in Fig. 5(b), the green and red dashed links represent the trees constructed for roots s_1 and s_2, respectively. The structurally aligned trees allow slice composition to be reduced to checking cut constraints on corresponding edges.

Let two D-slices be $D = (v_1, v_2, \ldots, v_k)$ and $D' = (v'_1, v'_2, \ldots, v'_k)$, where v_i and v'_i denote the reserved VM-slot capacities on server i in D and D', respectively. They correspond to two trees T and T' constructed for two different roots. If removing an edge set l in T and an edge set l' in T' separates the servers into the same two sets N_l and $\overline{N}_l$, then l and l' are called corresponding cut-edge sets. Each edge set may contain multiple edges, and the servers on either side need not form a connected component.

For a pair of corresponding edges l and l', define

$$H_D(l) = B \cdot \min\left\{\sum\nolimits_{i\in N_l} v_i, \sum\nolimits_{i\in \overline{N}_l} v_i\right\},$$
$$H_{D'}(l') = B \cdot \min\left\{\sum\nolimits_{i\in N_l} v'_i, \sum\nolimits_{i\in \overline{N}_l} v'_i\right\},$$
$$H_{D+D'}(l,l') = B \cdot \min\left\{\sum\nolimits_{i\in N_l} (v_i+v'_i), \sum\nolimits_{i\in \overline{N}_l} (v_i+v'_i)\right\}.$$

Here, $H_D(l)$ and $H_{D'}(l')$ denote the hose bandwidth demands of D and D' on the corresponding edges, while $H_{D+D'}(l,l')$ denotes the hose demand of the compound slice $D+D'$.

***Theorem** 5:* Within the tree-based composition model, two D-slices D and D' can be combined into a compound slice $D + D' = (v_1 + v'_1, v_2 + v'_2, \ldots, v_k + v'_k)$ if and only if the following condition holds for every pair of corresponding edges l and l':

$$H_{D+D'}(l,l') \le H_D(l) + H_{D'}(l').$$

Proof: We first prove necessity. Suppose there exists a pair of corresponding edges l and l' such that $H_{D+D'}(l,l') > H_D(l) + H_{D'}(l')$. Then, under the server partition defined by l and l', the compound slice may generate a worst-case cross-cut traffic demand larger than the total bandwidth provided by the two original D-slices on the corresponding edges. Hence, a valid hose traffic matrix can overload these cut-edge sets, making the compound slice infeasible.

We then prove sufficiency. Suppose that for every pair of corresponding edges l and l', we have $H_{D+D'}(l,l') \le H_D(l) + H_{D'}(l')$. For any valid hose traffic matrix of the compound slice, the traffic crossing each corresponding edge cut is upper-bounded by $H_{D+D'}(l,l')$. By the above condition, this value is no larger than the total bandwidth provided by D and D' on the corresponding edges. Therefore, no corresponding edge cut is overloaded. Within the tree-based composition model, every link bottleneck is characterized by its edge cut. Since the condition holds for all pairs of

TABLE II
ADDRESS DESCRIPTION

Address	Network Prefix (or Prefix)	Host Identifier (or ID)
1	192.168.0.0/16	1.100
2	203.55.196.0/24	130

corresponding edges, the compound slice $D + D'$ satisfies the hose bandwidth guarantee. ■

Consider two D-slice reservation vectors $D = (2, 5)$ and $D' = (3, 4)$. They can be combined according to Theorem 5, giving $D + D' = (5, 9)$. However, $D = (2, 3)$ and $D' = (3, 2)$ cannot be combined. Multiple slices can be composed into one compound slice through pairwise merging. Since both slice vectors and reserved bandwidth are additive, slice composition is associative. For example, when $D = (2, 0, 2, 2)$, $D' = (0, 0, 1, 1)$, and $D'' = (3, 2, 3, 2)$, all hierarchy-induced cut constraints are satisfied, and thus $D + D' + D'' = (D + D') + D'' = D + (D' + D'')$, yielding the compound slice $(5, 2, 6, 5)$.

Two incompatible D-slices can also be combined by "trimming" one slice so that Theorem 5 is satisfied. For example, if $D = (3, 5)$ and $D' = (4, 3)$ are incompatible, trimming D to $(3, 3)$ gives $(7, 6)$, while trimming D' to $(3, 3)$ gives $(6, 8)$. In both cases, the resulting compound slice is larger than either individual slice.

The composition of multiple D-slices can be extended similarly. For each group of corresponding edges, we check whether the worst-case cross-cut demand of the compound slice is no larger than the sum of the bandwidth demands of all involved slices on these edges.

B. Routing After Composition

Section IV has described the routing method for the single-root case. After composition, the multi-root case does not require a new routing model. Each VM-pair demand may be split among paths associated with different roots. Each allocated portion uses the source-side upstream and destination-side downstream reservations of its selected root-oriented path. Reservations from different slices sharing a physical link are accounted for together and must remain within that link's capacity in each direction. Therefore, the multi-root case reuses the single-root routing mechanism on the trees of different roots.

Specifically, the compound slice is jointly supported by the trees of multiple roots. The controller assigns traffic to a proper root according to the available bandwidth on the corresponding tree. Within the tree of a selected root, traffic is still forwarded according to the single-root routing method. Since Theorem 5 guarantees that the worst-case cross-cut demand on each group of corresponding edges does not exceed the total bandwidth provided by the involved slices, such multi-root routing does not violate the hose bandwidth guarantee.

VI. EXTENSIONS

This section considers a scale-out extension to a multi-level large-scale network, such as a metropolitan area network.

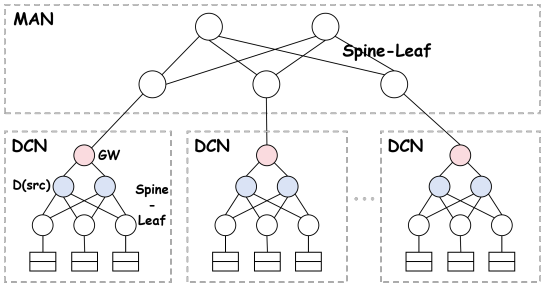


Fig. 6. Two-level MAN with each DCN as a virtual server.

A. MAN in a Two-Level Spine-Leaf

We demonstrate here that our approach can scale out to a large network setting with heterogeneous traffic between VMs. The scale-out network to be considered is a metropolitan area network [24]. Here, we can consider MAN that connects several DCNs. For example, each DCN in a MAN has a spine-leaf structure. In addition, all DCNs, treated as virtual servers, are connected through a global spine-leaf structure, where each DCN has a gateway serving as a leaf in the global spine-leaf. A server in a MAN is addressed in two parts: the network prefix (or simply prefix) and the host identifier (or simply ID) as shown in Table II. The prefix identifies the DCN, while the ID identifies a host within the DCN. We use n.prefix and n.ID to represent the DCN address and host ID within a DCN, respectively. Within a DCN, the gateway is connected to one or more switches, each is called a root.

In MAN, each VM has αB traffic bandwidth with other servers within the same DCN and has $(1 - \alpha)B$ traffic bandwidth with other servers outside the DCN. Hence, this is also called *mixed hose* model. We call traffic within a DCN east-west traffic determined by αB and the number of servers within a DCN. We call traffic between two DCNs north-south traffic. With the trend towards flattened communication, east-west traffic is gradually taking a dominant position [25]. Detailed network provisioning can be done as follows:

1) Construct the D-slice using the maximum flow at the supersource (gateway connected to multiple roots) at each DCN. $1 - \alpha$ amount of the resultant maximum flow will be counted as the capacity of a virtual server for the DCN at the MAN layer.
2) Construct the D-slice using the maximum flow in the MAN layer to obtain the flow assignment for each virtual server, representing a DCN.
3) Use the $1/(1-\alpha)$ amount of flow in each virtual server as the flow assignment to the corresponding gateway to all servers in the DCN.

Note that of all flows within a DCN, the α amount will be used for the intra-DCN flow, the $(1-\alpha)$ amount for the inter-DCN flow. This amount can be "traded" at the gateway for the same amount of intra-DCN flow. That is, $(1 - \alpha)$ for the up-phase from the sources to the gateway and the same amount for the down-phase from the gateway to the destinations.

Algorithm 5 Hierarchical MAN Routing

Input: source s and destination d
Output: A route from s to d

```
if s.Prefix = d.Prefix then
    Routing within the DCN
else
    Route from s to a root of the same DCN
    Send from the root to the GW of the DCN
    Route from the gateway of s to d's gateway through MAN based on d.Prefix.
    Send from d's gateway to a root of the destination DCN.
    Route from the root of d's DCN to d based on d's ID.
end if
```

B. Routing in MAN

Algorithm 5 shows a hierarchical two-level routing in a MAN as shown in Fig. 6 with a two-level spine-leaf structure, if s and d have the same prefix address, then routing is the same as the one within a DCN as shown in simple routing or short-cut routing. Otherwise, inter-DCN routing consists of three main steps: (1) moving out of s's DCN to its gateway (GW in Fig. 6), (2) routing within MAN from s' gateway to d' gateway through the MAN layer of DCNs, and (3) moving from d's gateway to destination d within the same DCN. This hierarchical design extends the intra-DCN routing mechanism to inter-DCN communication.

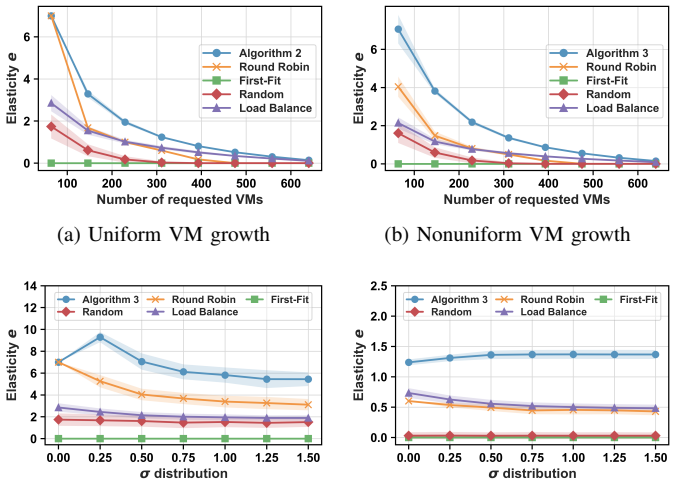
VII. Experimental Evaluation

We evaluated D-slice and short-cut routing in a large-scale simulation environment and a small-scale physical testbed.

A. Large-scale Simulation Evaluation

1) Experimental Settings: The simulation is built on Mininet [26], with network sizes varied as indicated in the corresponding figures. Mininet supports integration with SDN controllers and allows flexible configuration of network parameters such as bandwidth and latency. In terms of network architecture, we select a hierarchical Spine–Leaf (SL) topology as the structured topology [27], [28], and a Random Graph (RG) as the unstructured topology [29]–[31]. The SL simulation topology comprises three switch tiers: SuperSpine, Spine, and Leaf, with servers attached to Leaf switches. The RG generator first connects each new switch to a randomly selected existing switch to form a tree and supports adding randomly selected switch-to-switch edges to vary connectivity. Each server is attached to a randomly selected switch. Our simulation mainly focuses on the impact of network architecture, network size, and bandwidth on algorithm performance.

Link capacities are normalized by the bandwidth unit used in the model. Both normalized link capacities and server VM-slot capacities are sampled from Gaussian distributions with means ranging from 5 to 30 and standard deviations ranging from 1 to 3. Negative samples are set to zero, and the resulting values are rounded to integers. Bandwidth experiments increase link capacities from their initial values, with the increments shown on the horizontal axes.



(a) Uniform VM growth (b) Nonuniform VM growth

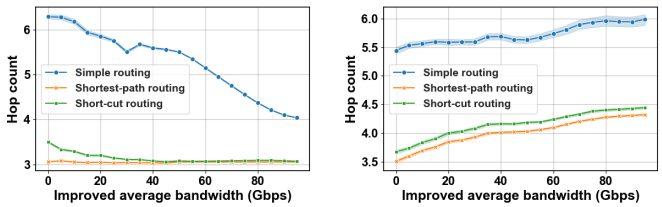
(c) Varying σ under minimum VMs (d) Varying σ under medium VMs

Fig. 7. Elastic performance with varying VMs.

To model application traffic patterns, we adopt the hose-model traffic matrix generation method in [32]. For each setting, 50 traffic matrices are generated, with each VM's aggregate outgoing and incoming traffic bounded by its bandwidth budget B. Each nonzero matrix is uniformly scaled until at least one VM's outgoing or incoming bandwidth bound is reached. All routing schemes use the same traffic matrices and VM placements, and the average communication hop count is reported. Since the same maximum single-root D-slice may correspond to multiple $D(src)$ and different roots affect routing performance, all possible $D(src)$ are traversed. We compare simple routing, short-cut routing, and shortest-path routing. Routing performance is evaluated using hop count, with outliers removed before averaging. Each experiment is repeated for 100 independent rounds; error bars show variation across rounds.

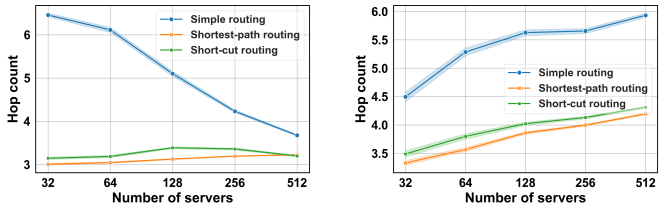
2) Elastic Performance with Varying VMs: We evaluate the computational elasticity of different VM placement strategies within a fixed D-slice reservation. Fig. 7(a) evaluates computational elasticity under uniform VM growth, where all VMs have the same growth coefficient. Elasticity decreases across all strategies as the number of VMs increases, reflecting the progressive reduction in available reserved capacity. The smallest reduction is observed for placements that preserve higher relative residual capacity after VM allocation, whereas strategies based solely on residual-slot counts or without explicit consideration of relative reservation exhibit lower elasticity. This difference becomes more pronounced as the system approaches capacity limits.

Fig. 7(b) shows computational elasticity under non-uniform VM growth, where growth coefficients are sampled from a truncated Gaussian distribution and normalized to have a unit mean. Across all evaluated application sizes, higher elasticity is observed when VM placements retain greater relative residual capacity after allocation. As reserved capacity becomes constrained, this difference becomes more pronounced. Unlike uniform growth, balancing VM counts or residual slots alone may concentrate high-growth VMs on individual servers, creating local growth bottlenecks. Accounting for the aggregate



(a) SL (b) RG

Fig. 8. Hop count with varying bandwidth in SL and RG.



(a) SL (b) RG

Fig. 9. Hop count with varying network size in SL and RG.

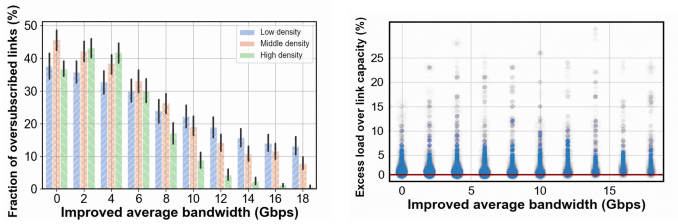
growth load on each server therefore provides an additional benefit in maintaining computational elasticity.

Figs. 7(c) and 7(d) further examine the effect of varying σ at the minimum and medium request sizes under the same truncated Gaussian distribution, with the mean growth coefficient fixed at unity. At the minimum request size, all growth coefficients are equal when $\sigma = 0$. Balancing VM counts then places one VM on each server, giving elasticity comparable to placements based on relative residual capacity. With a modest increase in σ, Algorithm 3 matches high-growth VMs to larger reservations, leaving less growth on the server with the smallest reservation. As σ increases further, however, each fast growing VM must remain on its host server, so unused capacity elsewhere cannot compensate, and elasticity falls. At the medium request size, more reserved slots are occupied, leaving less capacity for growth and lowering overall elasticity. Algorithm 3 balances this combined load against the remaining capacity, but the benefit of assigning smaller coefficients to the bottleneck server is less pronounced. Elasticity therefore rises smoothly and then levels off.

Taken together, these results show that preserving relative residual capacity is key to maintaining computational elasticity, while nonuniform growth further requires balancing the aggregate growth load across servers.

3) Hop Count under Varying Bandwidth and Network Size: We first evaluate the impact of link bandwidth, as shown in Fig. 8. In SL, the hop count of simple routing decreases as bandwidth increases, because higher link capacity provides flexibility in D-slice construction and allows more VM pairs to use shorter root-oriented paths. In contrast, short-cut routing remains nearly stable because it can avoid unnecessary traversal toward the root. In RG, the hop counts increase gradually with bandwidth. As the bandwidth constraint is relaxed, the D-slice can span a broader set of servers, including servers farther from the root and from each other, which slightly increases the average path length in the irregular topology.

We further evaluate the scalability of the routing schemes



(a) Fraction of oversubscribed links (b) Excess load over link capacity

Fig. 10. Shortest-path routing cannot provide network guarantee.

by varying the number of servers, as shown in Fig. 9. In SL, the hop count of simple routing decreases as the network size grows, whereas short-cut routing remains stable. This behavior reflects the scalability of the flat Spine–Leaf architecture: increasing the number of servers does not increase the number of network layers, so the path length remains bounded, while the larger network provides more placement choices that can reduce root-oriented detours. In RG, the hop counts increase with network size because enlarging the random topology generally increases the distance between communicating servers. Simple routing is more sensitive to this increase due to its mandatory traversal through the selected root, whereas short-cut routing can avoid unnecessary root traversal.

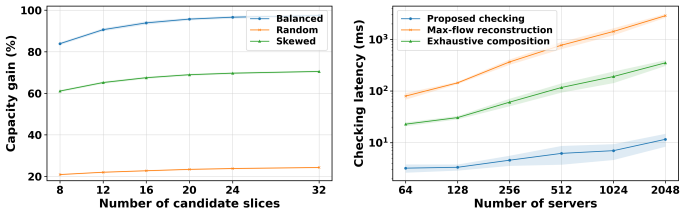
Overall, short-cut routing consistently achieves lower hop counts than simple routing under different bandwidth settings and network scales. Shortest-path routing generally achieves even lower hop counts, but its limitations in providing bandwidth guarantees are examined in the next subsection.

4) The Limitations of Traditional Routing Algorithms: Although shortest-path routing achieves lower hop counts in the preceding experiments, it is unaware of link bandwidth and the bandwidth reservations of D-slice, and thus cannot provide sufficient bandwidth guarantees. For example, routing 20 Gbps of traffic through a 10-Gbps link results in oversubscription, leading to congestion, high latency, and packet loss. In contrast, it follows from Theorem 4 that simple and short-cut routing do not exceed the reserved bandwidth of network links, ensuring that oversubscription does not occur.

We quantify this limitation using two metrics. The fraction of oversubscribed links is defined as the number of links whose traffic load exceeds their capacity divided by the total number of links. The excess load over link capacity measures how much the traffic load exceeds the capacity of an oversubscribed link, normalized by its link capacity. As shown in Fig. 10(a), more than 40% of links can become oversubscribed when the available bandwidth is limited, and this fraction decreases as the bandwidth increases. Fig. 10(b) further shows that some links carry more than 20% excess load over their capacities, with even higher values observed in some cases.

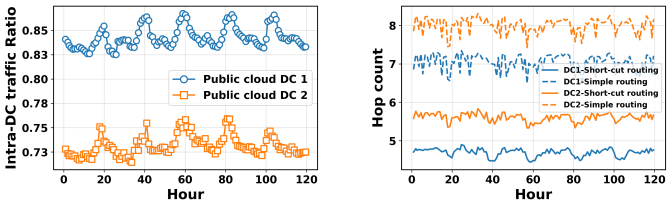
Evidently, under high traffic loads, such oversubscription is unacceptable, which is why shortest-path routing is not adopted as the routing algorithm in this work.

5) Slice Composition Performance: We evaluate slice composition on spine-leaf topologies using balanced, random, and skewed VM-slot distributions. For each instance, all candidate pairs are examined, and the feasible composition with the largest capacity is reported. As shown in Fig. 11(a),



(a) Capacity gain (b) Scalability of composition checking

Fig. 11. Slice composition performance in SL



(a) Traffic trace [33] (b) Hop count

Fig. 12. MAN traffic characteristics and routing performance under east-west and north-south traffic patterns.

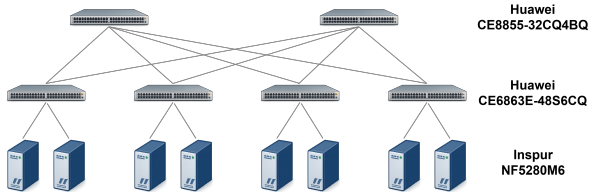
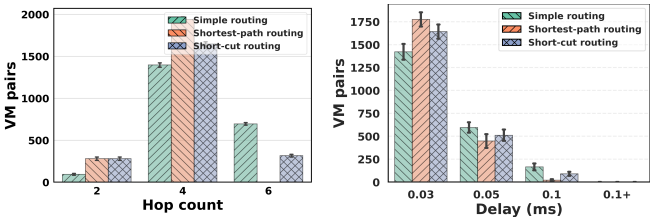


Fig. 13. Physical spine-leaf-server testbed topology.



(a) Hop count distribution (b) Delay distribution

Fig. 14. Testbed performance of short-cut routing.

the capacity gain increases with the number of candidates under all profiles. With 32 candidates, the balanced profile achieves the largest gain of approximately 97%, followed by the skewed profile at approximately 70% and the random profile at approximately 24%. Fig. 11(b) compares the proposed checking method with max-flow reconstruction, which rebuilds the compound flow network and reruns maximum flow, and exhaustive composition, which recomputes the cut demands for every candidate pair. With 2048 servers, the proposed method requires approximately 11.6 ms, whereas max-flow reconstruction and exhaustive composition require approximately 2.87 s and 351 ms, respectively.

Overall, slice composition provides substantial additional capacity, especially under balanced and skewed allocations. By reusing the cut information of existing slices, the proposed checking method searches for feasible compositions without repeated global reconstruction or redundant cut computation.

6) Performance in MAN: We evaluate the proposed routing schemes in a MAN scenario using traffic traces collected from two public cloud data centers over a continuous period of 120 hours [33]. As shown in Fig. 12(a), intra-DC traffic dominates, accounting for approximately 83%–86% in DC1 and 72%–76% in DC2, reflecting the prevalence of east-west communication in cloud workloads. Using these measured traffic ratios, we compare simple routing and short-cut routing in the two-level Spine–Leaf MAN topology.

Fig. 12(b) shows that short-cut routing consistently reduces the path length in both data centers: the average hop count decreases from about 7.0 to 4.7 in DC1 and from about 8.0 to 5.6 in DC2, corresponding to reductions of 33.4% and 30.0%, respectively. The slightly larger improvement in DC1 results from its higher fraction of intra-DC traffic, for which short-cut routing can more frequently bypass unnecessary traversal through the upper-level MAN structure. In contrast, inter-DC traffic requires traversing the MAN layer and therefore involves longer communication paths [34], [35]. The consistent reductions over time further demonstrate the effectiveness of short-cut routing under realistic and dynamically varying MAN traffic patterns.

Overall, these results show that short-cut routing remains effective in the two-level spine-leaf MAN architecture. It reduces hop count under realistic traffic distributions, while maintaining robust performance across different east-west and north-south traffic ratios.

B. Small-scale Testbed Evaluation

1) Experimental Settings: To verify the deployability of short-cut routing in real networks, we build a physical three-layer Spine–Leaf–Server testbed, as shown in Fig. 13. The testbed consists of two spine switches, four leaf switches, and eight servers. The servers are equipped with 32-core CPUs and 256 GB of memory. The spine layer uses 100-Gbps switches, while the leaf layer uses 25-Gbps switches. The following experiments are conducted on this testbed. Round-trip time (RTT) is used as the communication delay.

2) Short-cut Routing Performance: In this topology, communication between VMs attached to different servers under the same leaf switch requires two hops, while communication across different leaf switches requires four hops through the spine layer. When simple routing follows the root-oriented path, as illustrated by the blue route in Fig. 3, the path length can increase to six hops. Fig. 14(a) shows that short-cut routing assigns more traffic to shorter two-hop and four-hop paths and substantially reduces the fraction of six-hop paths compared with simple routing. On average, the hop count of short-cut routing is only 78% of that of simple routing. Shortest-path routing also produces short paths, but it is unaware of the hose-model bandwidth reservations and may therefore violate service-level guarantees under high load. Fig. 14(b) shows the corresponding delay distribution. By reducing unnecessary detours while preserving the D-slice bandwidth constraints, short-cut routing reduces the average latency by approximately 18% compared with simple routing.

The testbed results confirm the deployability of short-cut routing on real network devices and show that reducing

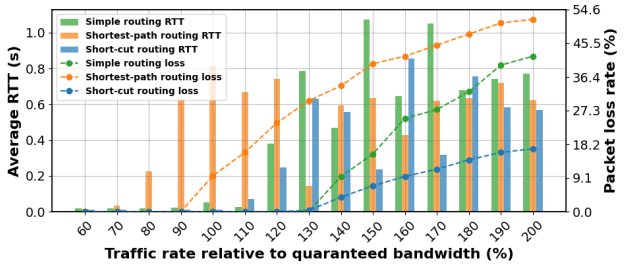


Fig. 15. Delay and packet loss rate with varying traffic.

unnecessary routing detours leads to lower hop count and end-to-end latency. These results are consistent with the simulation findings, further validating the effectiveness of short-cut routing in practical networks.

3) Bandwidth Guarantee and Burst Tolerance: D-slice reserves sufficient bandwidth to guarantee hose-model traffic, but this guarantee can be preserved only when routing follows the reserved bandwidth constraints. To evaluate both bandwidth guarantee and burst tolerance, we generate traffic matrices whose rates range from 60% to 200% of the guaranteed bandwidth and compare simple, shortest-path, and short-cut routing. As shown in Fig. 15, shortest-path routing begins to experience packet loss at around 100% of the guaranteed bandwidth, whereas simple and short-cut routing can tolerate 30% excess traffic without packet loss. Its RTT also increases sharply before reaching the guaranteed bandwidth, indicating early congestion. These results demonstrate that shortest-path routing may violate the D-slice bandwidth guarantee even when the offered traffic remains within the guaranteed range.

As the traffic load further increases, the difference in packet loss becomes more pronounced. At 200% of the guaranteed bandwidth, the packet loss rate of short-cut routing is only about 18%, compared with approximately 43% for simple routing and 52% for shortest-path routing. Notably, the packet loss of short-cut routing at 200% is comparable to that of shortest-path routing at only about 110%. Under such heavy loads, RTT should be interpreted together with packet loss: short-cut routing may exhibit a higher RTT because it successfully delivers a much larger fraction of the offered traffic, whereas simple and shortest-path routing discard substantially more packets.

These differences stem from their distinct routing behaviors. Shortest-path routing ignores the D-slice bandwidth reservations and may overload individual links, while simple routing preserves the bandwidth guarantee but consumes additional bandwidth through unnecessary root-oriented detours. Short-cut routing avoids such detours while respecting the reserved bandwidth constraints, thereby preserving the bandwidth guarantee of D-slice and retaining more residual network capacity to absorb traffic bursts.

VIII. Related Work

Related work in this area primarily spans hose-model-based resource provisioning, network slicing, and routing optimization in DCNs and MANs.

1) Hose-Based Resource Provisioning: The hose model was first proposed in [5] to simplify traffic specification by describing only the aggregate ingress/egress bandwidth of each endpoint. Subsequent studies further investigated hose-based VPN routing and provisioning, optimizing resource utilization and robustness under dynamic traffic [36], [37]. The hose model has also been applied to bandwidth guarantees in DCNs [38], [39] and network planning [32], [40] due to its flexible resource abstraction. Ahuja et al. [32] further studied a partial hose model that combines pipe and hose specifications.

Joint VM placement and network-resource allocation have been investigated through virtual-cluster abstractions [41], demand-aware placement to reduce bandwidth over-reservation [42], elasticity-aware placement [43], communication-cost-aware scaling [44], and hose-based multi-tenant placement [45]. More closely related work studied elastic scheduling under the hose model for trees [13] and general graphs [46]. In comparison, D-slice provides a topology-general resource structure that jointly specifies VM-slot capacity, hose-feasible bandwidth reservation, routing, and slice composition, with extensions to multi-root and multi-level networks with heterogeneous traffic.

2) Network Slicing and Resource Management: Network slicing [47] builds on earlier network virtualization research [48]. Studies on 5G slicing, NFV slicing, and cloud-network virtualization construct logically isolated resources on shared infrastructure, including virtual links, bandwidth partitions, and tenant isolation [49]–[53]. Related resource-management studies investigate bandwidth guarantees and fair sharing under variable traffic [54], [55]. Other studies examine elasticity management, prediction-driven scaling and routing, and dynamic resource scheduling in cloud-network environments [56]–[59]. Different from traditional virtual network slicing, the proposed D-slice jointly considers both computing and network resources under the hose model, including VM placement, bandwidth allocation, routing, and future traffic growth. It therefore serves as an elasticity-oriented cloud-network resource provisioning structure rather than a pure virtual network partition.

3) Routing in DCNs and MANs: Data center routing aims to improve bandwidth utilization, reduce delay, and alleviate congestion. Related techniques enable finer-grained traffic control through SDN, traffic engineering, and programmable switches [39], [60]. MAN routing focuses on high-bandwidth, low-latency connectivity between regional access networks and data centers [61], [62]. Traditional protocols such as OSPF [63] and IS-IS [64] maintain link-state information and routing tables. Our routing uses the reserved D-slice structure, reachable-destination information, and directional capacity states to reduce routing detours while preserving bandwidth feasibility.

IX. Discussion and Future Work

1) D-slice Trimming Strategies.: Making D-slices satisfy the composition conditions may require reducing reservations on different servers across multiple slices. The challenge is to jointly determine where and how much to trim: each

server participates in multiple bipartitions, so a local reduction affects several composition constraints, and different trimming choices incur different capacity losses. Trimming is therefore a combinatorial optimization problem with integer decisions and multiple cut constraints, aiming to maximize the total retained capacity while satisfying all composition constraints. This work provides feasible examples of enabling D-slice composition through trimming while retaining capacity benefits. More systematic trimming strategies and efficient algorithms remain directions for future research.

2) Heterogeneous VM Bandwidth Demands: A uniform bandwidth budget B allows link bandwidth to be expressed in terms of the number of VM slots it can support, bringing compute and network capacities into a common maximum-flow framework. With heterogeneous bandwidth demands, VMs occupy identical compute slots but require different amounts of bandwidth, so there is no uniform conversion between slot counts and bandwidth. The current construction can still provide an initial reservation based on B, followed by a reduction in the number of placed VMs according to their actual bandwidth demands and placement so that the reservations remain sufficient. However, this adjustment does not guarantee capacity optimality. Existing work on heterogeneous hose embedding has considered joint VM placement and routing [65]. Jointly optimizing compute and network reservations under heterogeneous bandwidth demands while supporting subsequent elastic growth is a promising direction for further research.

3) Extensions to Distributed AI Training: Distributed AI training provides a potential application of D-slice, where communication among workers involves collective operations rather than only individual point-to-point transfers. When workers are hosted in VMs, these operations can be represented by their constituent point-to-point transfers. The proposed routing preserves bandwidth feasibility provided that each VM's aggregate outgoing and incoming rates satisfy the hose bandwidth constraints. Beyond this bandwidth guarantee, further optimizing VM placement and routing based on the characteristics and traffic patterns of different collective operations is a direction for future research.

X. Conclusion

This paper presented a framework for elastic network planning in hose-based DCNs. The proposed approach is centered on D-slice, a DAG-shaped resource allocation structure constructed via maximum flow, which jointly determines VM-slot and bandwidth reservation to accommodate future traffic growth without re-scheduling. The framework supports simple state-light routing and a flow-guided short-cut routing scheme that improves bandwidth utilization while preserving hose bandwidth guarantees. We also studied the composition of multiple D-slices and established provable conditions for safe composition in multi-root topologies. Finally, the proposed design extends to multi-tier networks such as MANs and heterogeneous large-scale cloud-network infrastructures. Extensive simulation and testbed evaluations demonstrate its effectiveness in achieving high elasticity, efficient bandwidth utilization, and low routing delay.

References

[1] P. Kaushik, A. M. Rao, D. P. Singh, S. Vashisht, and S. Gupta, "Cloud computing and comparison based on service and performance between Amazon AWS, Microsoft Azure, and Google Cloud," in *Proc. of the International Conference on Technological Advancements and Innovations (ICTAI)*, 2021, pp. 268–273.

[2] China Telecom Corporation Limited, "Cloud-network integration 2035: Technical white paper," China Telecom Corporation Limited, Tech. Rep., 2025.

[3] S. Sen, R. D. Doverspike, and S. Cosares, "Network planning with random demand," *Telecommunication Systems*, vol. 3, no. 1, pp. 11–30, 1994.

[4] M. Gerla and L. Kleinrock, "On the topological design of distributed computer networks," *IEEE Transactions on Communications*, vol. 25, no. 1, pp. 48–60, 1977.

[5] N. G. Duffield, P. Goyal, A. Greenberg, P. Mishra, K. K. Ramakrishnan, and J. E. van der Merwe, "A flexible model for resource management in virtual private networks," *ACM SIGCOMM Computer Communication Review*, vol. 29, no. 4, pp. 95–108, 1999.

[6] D. Zhang, H. Wang, Y. Liu, X. Wei, Y. Shan, R. Chen, and H. Chen, "BlitzScale: Fast and live large model autoscaling with O(1) host caching," in *Proc. of the USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2025, pp. 275–293.

[7] Y. Xiang, X. Li, K. Qian, Y. Yang, D. Zhu, W. Yu, E. Zhai, X. Liu, X. Jin, and J. Zhou, "Aegaeon: Effective GPU pooling for concurrent LLM serving on the market," in *Proc. of the ACM SIGOPS Symposium on Operating Systems Principles (SOSP)*, 2025, pp. 1030–1045.

[8] D. Domingo, H. Barbalho, M. Molinaro, K. Liu, A. Pan, D. Dion, T. Moscibroda, S. Kannan, and I. Menache, "Kamino: Efficient VM allocation at scale with latency-driven cache-aware scheduling," in *Proc. of the USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2025, pp. 519–535. [Online]. Available: https://www.usenix.org/conference/osdi25/presentation/domingo

[9] C. Shou, G. Liu, H. Nie, H. Meng, Y. Zhou, Y. Jiang, W. Lv, Y. Xu, Y. Lu, Z. Chen, Y. Yu, Y. Shen, Y. Zhu, and D. Jiang, "InfiniteHBD: Building datacenter-scale high-bandwidth domain for LLM with optical circuit switching transceivers," in *Proc. of the ACM SIGCOMM Conference*, 2025, pp. 1–23.

[10] F. Gui, K. Gao, L. Chen, D. Li, V. Liu, R. Zhang, H. Yang, and D. Xiong, "Accelerating design space exploration for LLM training systems with multi-experiment parallel simulation," in *Proc. of the USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2025, pp. 473–488.

[11] K. Zhu, Y. Gao, Y. Zhao, L. Zhao, G. Zuo, Y. Gu, D. Xie, T. Tang, Q. Xu, Z. Ye, K. Kamahori, C.-Y. Lin, Z. Wang, S. Wang, A. Krishnamurthy, and B. Kasikci, "NanoFlow: Towards optimal large language model serving throughput," in *Proc. of the USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2025, pp. 749–765.

[12] Amazon Web Services, "Specifications for Amazon EC2 general purpose instances: Network performance," https://docs.aws.amazon.com/ec2/latest/instancetypes/gp.html, accessed: 2026-05-12.

[13] J. Wu, S. Lu, and H. Zheng, "On maximum elastic scheduling of virtual machines for cloud-based data center networks," in *Proc. of the IEEE International Conference on Communications (ICC)*, 2018, pp. 1–6.

[14] A. Kumar, R. Rastogi, A. Silberschatz, and B. Yener, "Algorithms for provisioning virtual private networks in the hose model," *IEEE/ACM Transactions on Networking*, vol. 10, no. 4, pp. 565–578, 2002.

[15] A. Gupta, J. Kleinberg, A. Kumar, R. Rastogi, and B. Yener, "Provisioning a virtual private network: a network design problem for multicommodity flow," in *Proc. of the 33rd Annual ACM Symposium on Theory of Computing (STOC)*, 2001, pp. 389–398.

[16] J. Edmonds and R. M. Karp, "Theoretical improvements in algorithmic efficiency for network flow problems," *Journal of the ACM*, vol. 19, no. 2, pp. 248–264, 1972.

[17] L. Chen, R. Kyng, Y. P. Liu, R. Peng, M. Probst Gutenberg, and S. Sachdeva, "Maximum flow and minimum-cost flow in almost-linear time," *Journal of the ACM*, vol. 72, no. 3, pp. 19:1–19:103, 2025.

[18] J. Wu, S. Quan, and X. Huang, "A comprehensive comparison of two resource allocation schemes in DCNs under the hose model," in *Proc. of the IEEE International Conference on Communications (ICC)*, 2026.

[19] M. Reitblatt, N. Foster, J. Rexford, C. Schlesinger, and D. Walker, "Abstractions for network update," in *Proc. of the ACM SIGCOMM Conference*, 2012, pp. 323–334.

[20] H. Stubbe, S. Gallenmüller, M. Simon, E. Hauser, D. Scholz, and G. Carle, "Exploring data plane updates on P4 switches with P4Runtime," *Computer Communications*, vol. 225, pp. 44–53, 2024.

[21] K. S. Namjoshi, S. Gheissi, and K. Sabnani, "Algorithms for in-place, consistent network update," in *Proc. of the ACM SIGCOMM Conference*. Association for Computing Machinery, 2024, pp. 244–257.

[22] The P4 Language Consortium, "P4-16 language specification," Version 1.2.5, Oct. 2024. [Online]. Available: https://p4.org/wp-content/uploads/sites/53/2024/10/P4-16-spec-v1.2.5.html

[23] N. erić, A. Varasteh, A. Blenk, and W. Kellerer, "NAGA: A deterministic programmable network with update timing guarantees," *IEEE Transactions on Network and Service Management*, vol. 22, no. 2, pp. 1874–1888, 2025.

[24] D. Sze, "A metropolitan area network," *IEEE Journal on Selected Areas in Communications*, vol. 3, no. 6, pp. 815–824, 1985.

[25] W. Xu, L. Zhang, and L. Chen, *Cloud Data Center Network Architecture and Technologies*, 2nd ed. Posts & Telecom Press, Jul. 2022, ch. 3.

[26] B. Lantz, B. Heller, and N. McKeown, "A network in a laptop: Rapid prototyping for software-defined networks," in *Proc. of the ACM SIGCOMM Workshop on Hot Topics in Networks (HotNets)*, 2010, pp. 1–6.

[27] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 4, pp. 63–74, 2008.

[28] A. Singh, J. Ong, A. Agarwal, G. Anderson, A. Armistead, R. Bannon, S. Boving, G. Desai, B. Felderman, P. Germano *et al.*, "Jupiter rising: A decade of Clos topologies and centralized control in Google's datacenter network," *ACM SIGCOMM Computer Communication Review*, vol. 45, no. 4, pp. 183–197, 2015.

[29] K. Qian, Y. Xi, J. Cao, J. Gao, Y. Xu, Y. Guan, B. Fu, X. Shi, F. Zhu, R. Miao, C. Wang, P. Wang, P. Zhang, X. Zeng, E. Ruan, Z. Yao, E. Zhai, and D. Cai, "Alibaba HPN: A data center network for large language model training," in *Proc. of the ACM SIGCOMM Conference*, 2024, pp. 691–706.

[30] H. H. Bazzaz, Y. Bi, W. Pang, M. Yu, R. Govindan, N. Cardwell, N. Dukkipati, M.-J. Tsai, C. DeForeest, Y. Jin, C. Carver, J. Kopański, L. Cheng, and A. Vahdat, "Preventing network bottlenecks: Accelerating datacenter services with hotspot-aware placement for compute and storage," in *Proc. of the USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2025, pp. 317–333. [Online]. Available: https://www.usenix.org/conference/nsdi25/presentation/bazzaz

[31] A. Valadarsky, G. Shahaf, M. Dinitz, and M. Schapira, "Xpander: Towards optimal-performance datacenters," in *Proc. of the ACM International Conference on Emerging Networking Experiments and Technologies (CoNEXT)*, 2016, pp. 205–219.

[32] S. S. Ahuja, V. Gupta, V. Dangui, S. Bali, A. Gopalan, H. Zhong, P. Lapukhov, Y. Xia, and Y. Zhang, "Capacity-efficient and uncertainty-resilient backbone network planning with hose," in *Proc. of the ACM SIGCOMM Conference*, 2021, pp. 547–559.

[33] F. Yan, C. Xie, J. Zhang, Y. Xi, Z. Yao, Y. Liu, X. Lin, J. Huang, Y. Ce, X. Zhang, and N. Calabretta, "Network traffic characteristics of hyperscale data centers in the era of cloud applications," *Journal of Optical Communications and Networking*, vol. 15, no. 10, pp. 736–749, 2023.

[34] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu *et al.*, "B4: Experience with a globally-deployed software defined WAN," *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 4, pp. 3–14, 2013.

[35] N. H. Rotman, Y. Ben-Itzhak, A. Bergman, I. Cidon, I. Golikov, A. Markuze, and E. Zohar, "CloudCast: Characterizing public clouds connectivity," *arXiv preprint arXiv:2201.06989*, 2022.

[36] M. Kodialam, T. V. Lakshman, and S. Sengupta, "Traffic-oblivious routing in the hose model," *IEEE/ACM Transactions on Networking*, vol. 19, no. 3, pp. 774–787, 2011.

[37] A. Jüttner, I. Szabó, and Á. Szentesi, "On bandwidth efficiency of the hose resource management model in virtual private networks," in *Proc. of the IEEE Conference on Computer Communications (INFOCOM)*, vol. 1, 2003, pp. 386–395.

[38] L. Popa, P. Yalagandula, S. Banerjee, J. C. Mogul, Y. Turner, and J. R. Santos, "ElasticSwitch: Practical work-conserving bandwidth guarantees for cloud computing," in *Proc. of the ACM SIGCOMM Conference*, 2013, pp. 351–362.

[39] S. Wang, K. Gao, K. Qian, D. Li, R. Miao, B. Li, Y. Zhou, E. Zhai, C. Sun, J. Gao, D. Zhang, B. Fu, F. Kelly, D. Cai, H. H. Liu, and M. Zhang, "Predictable vFabric on informative data plane," in *Proc. of the ACM SIGCOMM Conference*, 2022, pp. 615–632.

[40] J. P. Eason, X. He, R. Cziva, M. Noormohammadpour, S. Balasubramanian, S. S. Ahuja, and B. Lu, "Hose-based cross-layer backbone network design with Benders decomposition," in *Proc. of the ACM SIGCOMM Conference*, 2023, pp. 333–345.

[41] H. Ballani, P. Costa, T. Karagiannis, and A. Rowstron, "Towards predictable datacenter networks," in *Proc. of the ACM SIGCOMM Conference*, 2011, pp. 242–253.

[42] J. Lee, Y. Turner, M. Lee, L. Popa, S. Banerjee, J.-M. Kang, and P. Sharma, "Application-driven bandwidth guarantees in datacenters," in *Proc. of the ACM SIGCOMM Conference*, 2014, pp. 467–478.

[43] K. Li, J. Wu, and A. Blaisse, "Elasticity-aware virtual machine placement for cloud datacenters," in *Proc. of the IEEE International Conference on Cloud Networking (CloudNet)*, 2013, pp. 99–107.

[44] S. Lu, Z. Fang, J. Wu, and G. Qu, "Elastic scaling of virtual clusters in cloud data center networks," in *Proc. of the IEEE International Performance Computing and Communications Conference (IPCCC)*, 2017, pp. 1–8.

[45] S. Lu, J. Wu, and Z. Fang, "High-elasticity virtual cluster placement in multi-tenant cloud data centers," in *Proc. of the IEEE International Conference on High Performance Computing and Communications, IEEE International Conference on Smart City, and IEEE International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*, 2019, pp. 996–1003.

[46] Y. Qwareeq, A. Sawwan, and J. Wu, "Maximum elastic scheduling of virtual machines in general graph cloud data center networks," *Cyber-Physical Systems*, vol. 10, no. 3, pp. 283–301, 2024.

[47] Y. Yiakoumis, K.-K. Yap, S. Katti, G. Parulkar, and N. McKeown, "Slicing home networks," in *Proc. of the ACM SIGCOMM Workshop on Home Networks*, 2011, pp. 1–6.

[48] A. Bavier, N. Feamster, M. Huang, L. Peterson, and J. Rexford, "In VINI veritas: realistic and controlled network experimentation," in *Proc. of the ACM SIGCOMM Conference*, 2006, pp. 3–14.

[49] F. Fossati, S. Moretti, P. Perny, and S. Secci, "Multi-resource allocation for network slicing," *IEEE/ACM Transactions on Networking*, vol. 28, no. 3, pp. 1311–1324, 2020.

[50] Q.-T. Luu, S. Kerboeuf, A. Mouradian, and M. Kieffer, "A coverage-aware resource provisioning method for network slicing," *IEEE/ACM Transactions on Networking*, vol. 28, no. 6, pp. 2393–2406, 2020.

[51] Y. Zhou, M. Wilkening, J. Mickens, and M. Yu, "SmartNIC security isolation in the cloud with S-NIC," in *Proc. of the European Conference on Computer Systems (EuroSys)*, 2024, pp. 851–869.

[52] L. Peterson, T. Anderson, S. Katti, N. McKeown, G. Parulkar, J. Rexford, M. Satyanarayanan, O. Sunay, and A. Vahdat, "Democratizing the network edge," *ACM SIGCOMM Computer Communication Review*, vol. 49, no. 2, pp. 31–36, 2019.

[53] J. Dai, L. Li, R. Safavinejad, S. Mahboob, H. Chen, V. V. Ratnam, H. Wang, J. Zhang, and L. Liu, "O-RAN-enabled intelligent network slicing to meet service-level agreement (SLA)," *IEEE Transactions on Mobile Computing*, vol. 24, no. 2, pp. 890–906, 2025.

[54] F. Liu, J. Guo, X. Huang, and J. C. S. Lui, "eBA: Efficient bandwidth guarantee under traffic variability in datacenters," *IEEE/ACM Transactions on Networking*, vol. 25, no. 1, pp. 506–519, 2017.

[55] J. Guo, F. Liu, J. C. S. Lui, and H. Jin, "Fair network bandwidth allocation in IaaS datacenters via a cooperative game approach," *IEEE/ACM Transactions on Networking*, vol. 24, no. 2, pp. 873–886, 2016.

[56] Q.-T. Luu, S. Kerboeuf, and M. Kieffer, "Uncertainty-aware resource provisioning for network slicing," *IEEE Transactions on Network and Service Management*, vol. 18, no. 1, pp. 79–93, 2021.

[57] A. A. Abdellatif, A. Mohamed, A. Erbad, and M. Guizani, "Dynamic network slicing and resource allocation for 5G-and-beyond networks," in *Proc. of the IEEE Wireless Communications and Networking Conference (WCNC)*, 2022, pp. 262–267.

[58] Y. Zheng, L. Hua, Y. Li, and H. Lu, "Adaptive network slicing and hierarchical data management for edge-assisted multi-task video enhancement," *IEEE Network*, vol. 39, no. 3, pp. 63–71, 2025.

[59] X. Fei, F. Liu, H. Xu, and H. Jin, "Adaptive VNF scaling and flow routing with proactive demand prediction," in *Proc. of the IEEE Conference on Computer Communications (INFOCOM)*. IEEE, 2018, pp. 486–494.

[60] B. Chang, K. He, S. S. Chen, J. Lin, M. Zhang, W. Wu, and A. Akella, "Balancing SDN control plane availability and traffic engineering efficiency in data centers," in *Proc. of the IEEE International Conference on Network Protocols (ICNP)*, 2024, pp. 1–12.

[61] M. Markovitch and S. Schmid, "SHEAR: A highly available and flexible network architecture marrying distributed and logically centralized control planes," in *Proc. of the IEEE International Conference on Network Protocols (ICNP)*, 2015, pp. 78–89.

[62] M. Zhou, X. Zhang, S. Hao, X. Yang, J. Zheng, G. Chen, and W. Dou, "Regional IP anycast: deployments, performance, and potentials," in *Proc. of the ACM SIGCOMM Conference*, 2023, pp. 917–931.
[63] J. Moy, "OSPF version 2," RFC 2328, Internet Engineering Task Force, Apr. 1998, standards Track. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc2328
[64] R. Callon, "Use of OSI IS-IS for routing in TCP/IP and dual environments," RFC 1142, Internet Engineering Task Force, Feb. 1990, informational. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc1142
[65] F. Yan, T. T. Lee, and W. Hu, "Congestion-aware embedding of heterogeneous bandwidth virtual data centers with hose model abstraction," *IEEE/ACM Transactions on Networking*, vol. 25, no. 2, pp. 806–819, Apr. 2017.



Jie Wu (Fellow, IEEE) is Laura H. Carnell Professor at Temple University and the Director of the Center for Networked Computing (CNC). His current research interests include mobile computing and wireless networks, routing protocols, network trust and security, distributed algorithms, applied machine learning, and cloud computing. He serves on several editorial boards, including IEEE/ACM Transactions on Networking. Dr. Wu was general chair/co-chair for IEEE IPDPS'23, ACM MobiHoc'23, and IEEE CCGrid'24 as well as program chair/cochair for IEEE INFOCOM'11 and CCF CNCC'13. He was an IEEE Computer Society Distinguished Visitor, ACM Distinguished Speaker, and chair for the IEEE Technical Committee on Distributed Processing (TCDP). Dr. Wu is a Fellow of the AAAS and a Fellow of the IEEE. He is the recipient of the 2011 China Computer Federation (CCF) Overseas Outstanding Achievement Award. He is a Member of the Academia Europaea (MAE). Currently, Dr. Wu is on leave working at China Telecom as a Scientist in Cloud Computing.



Shuo Quan (Member, IEEE) received the B.S. degree in Software Engineering from Beijing University of Posts and Telecommunications in 2013 and master's degree in Software Engineering from Beijing University of Posts and Telecommunications in 2016. He is a researcher at China Telecom Cloud Computing Research Institute, Beijing, China. His research interests include cloud computing, networking and communication systems, cloud network convergence, cloud-native and virtualization, etc.



Xiaoyao Huang (Member, IEEE) received the B.S. degree from Dalian Maritime University, Dalian, China, in 2015, and the Ph.D. degree from the University of Chinese Academy of Sciences, Beijing, China, in 2022. She is currently a Researcher with the China Telecom Cloud Computing Research Institute, Beijing, China. Her research interests include computing power networks, cloud-edge collaborative computing, task scheduling and resource orchestration in large-scale distributed systems, and AI-driven networking systems.



Huitao Zhou (Member, IEEE) received the B.S. degree from Dalian National University, Dalian, China, in 2021. He is currently working toward the Ph.D. degree in information and communication engineering with Beijing University of Posts and Telecommunications, Beijing, China. His research interests include data center networks, intelligent computing interconnection networks, and cloud network convergence.