Efficient and Green Embedding of Virtual Data Centers with Mixture of Unicast and Multicast Services

Chao Guo, Kai Xu, Sanjay Kumar Bose, Senior Member, IEEE, Moshe Zukerman, Fellow, IEEE, and Gangxiang Shen, Senior Member, IEEE

Abstract—The improved efficiency achieved by virtualizing data centers (DCs) has been well established. In this paper, we propose a mixed Virtual Data Center (VDC) capable of supporting both unicast and multicast services. We provide a new method to realize the embedding of these VDCs. We also provide a Mixed Integer Linear Programming (MILP) formulation and a scalable heuristic algorithm for efficiently embedding its demands. Numerical results show that mixed VDC embedding supporting both unicast and multicast services performs significantly better than existing embedding methods in terms of system cost, power consumption, link capacity utilization, and VDC acceptance ratio.

Index Terms—Data center, virtual data center, multicast, unicast

1 INTRODUCTION

CLOUD computing has gained significant popularity in cost-effectively supporting new business models such as Infrastructure-as-a-Service (IaaS), Platform-as-a-Service (PaaS), and Software-as-a-Service (SaaS) [1]. However, current cloud providers mainly offer resources in terms of virtual machines (VMs), where the temporal computing-intensive peaks in one VM may disturb other VMs. If such cloud providers do not design their networks to specifically guarantee performance isolation, then this may adversely affect the performance of the deployed services and the DC resource utilization [2]. To address these weaknesses, new architectures like SecondNet [3], Octopused [4], and NetLord [5] have been proposed by fully virtualizing data centers. A virtual data center (VDC) is composed of a set of VMs with assigned computing and storage resources, connected by virtual links with assigned capacity. In this way, infrastructure providers (InPs) like Amazon EC2 [6], Google App Engine [7], and Microsoft Azure [8] can create multiple VDCs on top of shared physical DCs, and then rent these VDCs to service providers (SPs). After the latter obtain one or more VDCs, they can provide cloud services to their end users [2].

To enable VDCs to efficiently share common DC resources spread over many physical nodes and links, VDC embedding is an essential but challenging research problem. Although VDC embedding has some similarity to virtual network embedding (VNE), the former has to overcome a significantly higher complexity because of the large number of nodes in a DC and its complex features [9]. Prior research on VDC embedding has focused on performance criteria such as system cost [10], energy consumption [11], and reliability [12] and have mainly used them to evaluate the performance of unicast services. Many applications in a DC may actually generate a mixture of unicast and multicast traffic. For example, the Hadoop framework consists of two main parts - the MapReduce process and the Hadoop Distributed File System (HDFS) [13]. While a large amount of unicast traffic is there between mappers and reducers [13], multicast traffic also exists during the shuffle stage of the MapReduce procedure and in the HDFS [14]. Using unicast mode alone to also deliver multicast traffic would waste a lot of transmission capacity which would in turn lead to a higher cost (i.e., energy cost and hardware cost) and/or possible violation of quality of service (QoS) requirements (i.e., higher service rejection ratios and/or excessive delays).

To overcome these drawbacks, this paper proposes to incorporate native multicasting capability in a DC to configure VDCs that can support a mixture of unicast and multicast services. Although multicast embedding has been considered in [15], [16], and [17], these studies focus on backbone networks or wireless mesh networks. Also, although the research in [18], [19], and [20] aims to improve efficiency and service quality of data center networks (DCNs), these do not consider VDCs, nor do they consider supporting both unicast and multicast services in the same VDC. The key novelty and contributions of this paper are as follows.

- Propose the concept of mixed VDC embedding: In a mixed VDC, both unicast virtual links and multicast (virtual) trees are present. A multicast tree in a VDC is composed of a group of virtual links connecting one source node to all the other destination nodes in the tree. Unlike unicast virtual links, all the virtual links in the virtual tree are unidirectional. Moreover, the
capacities of the virtual links in the tree are equal to each other as they are responsible for delivering the same data from the source node to all the destination nodes in the tree. Also, when embedding a mixed VDC on a substrate DCN, we fully utilize the native multicast capability of the switch elements to efficiently support multicast services in the VDC.

- **Develop a MILP optimization model for mixed VDC embedding which embeds multicast virtual links in a multicast approach:** The primary objective of the MILP model is to minimize system cost and energy consumption while improving link capacity utilization. This model considers static VDC demands where for a given set of VDC requests, their arrival and holding times and resource requirements are known in advance. This, for example, applies to situations where tenants pre-order VDC services for some time in the future.

- **Develop an efficient heuristic algorithm to embed mixed VDC in a multicast approach aiming to achieve the same goal as the MILP model:** This heuristic algorithm has significantly lower time complexity than the MILP model and can therefore implement mixed VDC embedding even for large, realistically sized networks and for network with dynamic service requests.

- **Simulation studies for scenarios involving both static and dynamic VDC demand:** For the static demand, we employ both the MILP model and the heuristic algorithm to implement mixed VDC embedding. Numerical results for small size problems demonstrate the efficiency of the proposed heuristic algorithm, which performs close to the MILP model. The numerical results also show that the proposed heuristic algorithm based on mixed VDC embedding significantly outperforms a heuristic algorithm where only unicast services are supported (i.e., the algorithm with unicast embedding). We have a similar observation on the performance of the proposed heuristic algorithm under dynamic demands, where the algorithm considering mixed VDC embedding achieves significantly better performance in terms of VDC rejection ratio than both the algorithm with unicast embedding and the approach employed in the SecondNet framework.

The remainder of this paper is organized as follows. In Section 2, we survey existing research on VDC embedding. We then introduce the concept of mixed VDC embedding in Section 3. In Section 4, we define the problem of mixed VDC embedding, based on which we further present its MILP model. An efficient heuristic algorithm on how to efficiently embed mixed unicast and multicast services is presented in Section 5. We evaluate the performance of the proposed schemes through simulations in Section 6. Section 7 concludes the paper.

## 2 RELATED WORK

### 2.1 VDC Embedding

The VDC embedding problem is to map required VDC resources such as VMs and virtual link capacities onto physical servers and links. The existing research on VDC embedding has been limited to being mainly focused on the following directions.

Several publications aim at minimizing energy consumption, in addition to minimizing system cost, through efficient resource mapping. Nam et al. [1] proposed switching off idle servers to reduce power consumption and proposed embedding VMs as close to each other as possible to minimize the communication required. In [11], Ghasizadeh et al. considered reducing power consumption when embedding MapReduce-based virtual networks onto a DCN. Further, Amokrane et al. [21] proposed reducing power consumption by embedding VDCs onto several different DCs. Yang et al. [22] and Chang et al. [23] considered virtual switches in VDCs and selected servers as close as possible while embedding VMs and mapping virtual switches and links simultaneously so as to reduce both energy consumption and system cost. Zhani et al. [10] proposed reducing energy consumption and system cost by migrating VMs.

There are also studies on reliability. Zhang et al. [12] proved that the problem of reliable VDC embedding is NP-hard and proposed a framework called Venice to address the problem. Wen et al. [9] developed a reliable VDC embedding framework based on software-defined networking (SDN). Sun et al. [24] proposed a heuristic algorithm to embed reliable VDCs across multiple DCNs. Yu et al. [25] considered survivability for the VDC embedding problem. Lo et al. [26] proved that the problem of survivable VDC allocation is NP-hard and proposed a framework called CALM to provide a solution that recovers switch failures.

Improving the successful VDC embedding ratio has been another important objective of VDC embedding. In [27], Chowdhury et al. implemented joint embedding for both virtual nodes and links to improve the successful VDC embedding ratio. In [28], Rabbani et al. developed an approach to jointly embed VMs onto servers and virtual switches onto physical switches. In [29], Gilesh et al. improved the successful embedding ratio using an approach that can minimize the fragmentation of residual resource by migrating VMs appropriately.

There are also studies focusing on security [30] and network topology [31]. However, almost all the existing publications on VDC embedding deal only with unicast services and have not considered embedding VDCs with a mixture of both unicast and multicast services.

### 2.2 Multicast in DCNs

Data centers today have to support a wide variety of services, of which many are inherently multicast. For example, distributed file systems such as the Google File System (GFS) and the Hadoop Distributed File System (HDFS) use multicasting to deliver the same information from one source node to many destination nodes [14]. Parallel database join operations [32], VM provisioning [33], and in-cluster software updating [34] also use multicasting. Therefore, it is important to consider multicasting requirements when embedding VDCs.
Several studies have indeed looked into multicast in DCs without considering VDC embedding. Samadi et al. [14] proposed a multicast system for DCNs based on the multicast capability of optical components such as optical splitters. In [35], Li et al. presented efficient and scalable multicast routing algorithms for DCNs. In [36], Saridis et al. presented an all-optical DCN that dynamically offers the multicast capability for VDC applications. However, these studies have only focused on a DCN, but not on VDC embedding.

2.3 Summary
In summary, the existing studies on VDC embedding have not considered situations where there is a mixture of unicast and multicast services. Also, although there are studies considering multicast in DCs, these do not focus on the problem of VDC embedding. However, it would be realistic to have a VDC today that can support a mixture of both unicast and multicast services. Therefore, it is important to develop effective approaches for VDC embedding that would support such a mixture of services. This work contributes to this objective by providing efficient approaches for mixed VDC embedding.

3 Mixed VDC Embedding Supporting Both Unicast and Multicast Services
In this section, we first introduce a physical DC, then describe how to realize multicast in a DCN, and finally introduce the concept of mixed VDC embedding supporting both unicast and multicast services.

3.1 Data Centers with VDC Embedding
A physical DC is typically organized as a tree-like topology in three tiers as shown in Fig. 1. Servers are placed in racks and each server is directly connected to a switch on the top of the rack (i.e., Top-of-Rack (ToR) switch). The ToR switches are further connected to several second-tier switches (i.e., aggregation switches). The latter are further connected to core switches. Optical fibers are only used to interconnect DCN switches, and the links between servers and ToR switches are often made of copper (e.g., Ethernet cables) [37], [38]. In the context of this network layout, the multicast capability can be achieved both in the electronic domain by deploying Internet Protocol (IP multicast), as well as in the optical domain by using passive optical splitters [14].

The concept of VDC embedding is to use a DC to support multiple VDCs by allowing them to share the common resources of the DC virtually. Fig. 1 illustrates an example involving two VDCs, each of which requires computing and storage resources at a VDC node (i.e., VM node) and capacity on a VDC link (i.e., virtual link). Here we are adopting the same definition of a VDC as in, e.g., [1], [3], [9], where no virtual switches are considered in the context of a VDC. There are also publications that consider virtual switches for VDCs, e.g., [22] and [28]. In our opinion, the difference between the two definitions of a VDC are not crucial, but they do differ on whether to explicitly allocate switching capability to each VDC. With bandwidth guaranteed for each virtual link of a VDC as in [1], [3], and [9], the VDC essentially possesses virtual switching capability decomposed from physical switches since each virtual link traverses multiple physical switches. The guarantee of virtual link bandwidth means the assurance of allocated switching capability at these physical switches.

In the example of Fig. 1, all virtual resources in VDCs are provided by the physical resources in the DC through a specific mapping relationship. The communication between the VM nodes in a VDC is carried out through either unicast or multicast. In Fig. 1, the unicast traffic is transmitted over unicast virtual links (shown by solid lines), which are bidirectional, while the multicast traffic is delivered by multicast trees (shown by dotted lines), each of which is composed of a group of unidirectional virtual links with the same source node and different destination nodes. In addition, all the virtual links on a multicast tree have the same capacity as they transmit the same information from the source node to all the destination nodes in the tree. As an example, VDC2 in Fig. 1 consists of three unicast virtual links and one multicast tree.

3.2 Mapping Multicast Trees for VDCs
Fig. 2 shows an example of mapping a multicast tree for a VDC where the multicast tree is over four VM nodes and three virtual links. Here VM0 is the source node, and VM1, VM2, and VM3 are the destination nodes. The computing resource (in units of CPUs) required by each VM node is shown in the figure. Suppose that the given servers have sufficient computing resources, and the VM nodes VM0, VM1, VM2, and VM3 are mapped onto servers 0, 1, 2, and 3, respectively. Also, assume that the capacity of each virtual link in the tree is one unit. Then we can have the following two approaches to map these virtual links.

Fig. 2(b) illustrates the multicast approach, which relies on the multicast capabilities in physical switches to support multicast services. Under this approach, a multicast tree can be built with only one unit of link capacity reserved at each physical link traversed by the tree (e.g., Link 4-1). This is because the physical switches traversed by the tree, e.g., Node 4 (switch), can duplicate the information sent from Node 0 (source), and then transmit it to the next physical nodes. Also, for data delivery, only one copy of data processing capacity is required at Node 0.
embedding to support multicast services may cause significant wastage of both link capacity and server resources. This wastage may lead to a higher system cost because more servers and links have to be used. In addition, such wastage will cause an increase in power consumption because more equipment has to be deployed and turned on. To avoid this, we propose to employ mixed VDC embedding, where while a unicast communication technique is applied to deliver a unicast traffic, the multicast-enabled VDC embedding, shown in Fig. 2(b), is employed for the multicast traffic based on the concept of mixed VDC embedding. Next, we introduce the problem definition and then the MILP optimization model and the proposed heuristic algorithm.

4.1 Problem Definition

The problem of mixed VDC embedding is defined as follows. We are given a physical DC as the one shown in Fig. 1, which consists of a set of server nodes, a set of switch nodes, and a set of physical links interconnecting these server nodes and switch nodes. Each server node provides resources of CPU, memory, and disk space. The physical links between server nodes and ToR switch nodes are copper based (Ethernet cables), while the other links are optical fiber cables. All switches in the DCN are assumed to have multicast capability.

Each VDC request consists of a set of VM nodes, each of which needs resources of CPU, memory, and disk space from a server node. Each VDC may include multiple unicast virtual links, which are used for delivering unicast traffic and are bidirectional. Multicast trees are also configured in each VDC, where the virtual links in each tree are used for delivering multicast traffic and are all unidirectional (see Fig. 1). The capacity required by each virtual link of a multicast tree is the same, and only one unit of such capacity is reserved on a physical link commonly traversed by these virtual links (as shown in Fig. 2(b)).

We call both unicast virtual links and multicast trees sessions. For example, a multicast tree in a VDC is a session with one source node and several destination nodes. Although a unicast session can be considered as a special case of a multicast session, we distinguish them by using two different demand matrices.

We consider VDC embedding scenarios involving both static and dynamic VDC requests, which are introduced as follows.

4.1.1 Static Scenario

Under the static scenario, all the VDC requests are given in advance. The arrival time and duration of each VDC are also known in advance. This applies to situations where tenants order VDC services for use at some time in the future. In this case, we aim to minimize physical resources in terms of servers and link capacity used subject to the condition that all the VDCs are embedded onto the physical infrastructure. For this scenario, we formulate the problem as a MILP model that is applicable to embedding VDCs with both unicast and multicast services. We also propose a heuristic algorithm for solving this embedding problem.

4 Problem Formulation and MILP Model for Mixed VDC Embedding

As shown by the example in Section 3.2, using unicast VDC
4.1.2 Dynamic Scenario

Under the dynamic scenario, VDC requests arrive randomly and sequentially. These are then established for a certain period until they finally depart. Here, due to the limitation of physical resources, some requests may be rejected. Therefore, for this scenario, we aim to minimize the VDC rejection ratio (i.e., the ratio of the total number of rejected requests out of the total number of requests that arrive during the period over which the system is observed), as this would improve the revenue of an InP.

4.2 MILP Model

For the static scenario, we develop a MILP model to embed VDCs optimally. Here for building a multicast tree, we refer to the work in [39] when formulating our MILP model. The MILP model is given below.

Sets:
- \( P \) - Set of server nodes.
- \( S \) - Set of switch nodes, including ToR, aggregation, and core switches.
- \( NE_m \) - Set of neighbor nodes of node \( m \). Here a node can be a server node or a switch node.
- \( R \) - Set of server resource types, including CPU, memory, and disk space.
- \( \Omega \) - Set of VDC requests.
- \( T \) - A time sequence or set of times, each of which corresponds to the arrival time of each considered VDC request.
- \( N_i \) - Set of VM nodes in VDC \( i \).
- \( Y_i \) - Set of multicast sessions in VDC \( i \).
- \( \Theta_{ij} \) - Set of VM nodes in multicast session \( j \) of VDC \( i \), represented as \([s_{ij}, d^1_{ij}, d^2_{ij}, \ldots, d^{|Y_i|}_{ij}]\), where \( s_{ij} \) is the source VM node and \( d^1_{ij}, d^2_{ij}, \ldots, d^{|Y_i|}_{ij} \) are the destination VM nodes of the session.

Parameters:
- \( C_r \) - Capacity of resource type \( r \) in each server node. We assume that each server has the same resources.
- \( B_{mn} \) - Capacity of physical link \((m, n)\), where \( m \) and \( n \) are the two end nodes of the link.
- \( t_i \) - The arrival time of VDC request \( i \).
- \( \Delta t_i \) - Duration of VDC \( i \).
- \( \beta_{it} \) - A binary parameter to denote whether VDC \( i \) is still active at time \( t \). Specifically, we have \( \beta_{it} = \begin{cases} 1 & t_i \leq t < t_i + \Delta t_i \\ 0 & \text{otherwise} \end{cases} \forall i \in \Omega \)
- \( RD^r_{v} \) - Amount of resource type \( r \) required by VM node \( v \) in VDC \( i \).
- \( L_{ij} \) - Number of VM nodes in multicast session \( j \) of VDC \( i \).
- \( \lambda_{ij} \) - Virtual link capacity required by multicast session \( j \) in VDC \( i \). Here each virtual link is assumed to have the same amount of capacity.
- \( \lambda^{sd}_{mn} \) - Virtual link capacity required by the unicast session between VM node pair \((s, d)\) of VDC \( i \). We assume that there are two opposite unicast sessions between each VM node pair.

Variables:
- \( \delta^l_{vm} \) - A binary variable that takes the value of 1 if VM node \( v \) of VDC \( i \) is embedded onto server node \( n \); 0, otherwise.
- \( \omega_n \) - A binary variable that takes the value of 1 if server node \( n \) is used; 0, otherwise.
- \( \sigma_{im} \) - A binary variable that takes the value of 1 if the tree of multicast session \( j \) in VDC \( i \) uses physical node \( m \); 0, otherwise.
- \( \mu_{mn} \) - A binary variable that takes the value of 1 if the tree of multicast session \( j \) in VDC \( i \) traverses physical link \((m, n)\); 0, otherwise.
- \( \varphi_{ij} \) - An integer variable that denotes the number of commodity flows of multicast session \( j \) in VDC \( i \) that traverse physical link \((m, n)\).
- \( \tau^i_{n} \) - A binary variable that takes the value of 1 if only the destination VM nodes of multicast session \( j \) in VDC \( i \) are mapped onto server node \( n \), but its source VM node is not mapped onto this server node; 0, otherwise. We call this type of server node destination server node of session \( j \) in VDC \( i \), and it receives data only from this session.
- \( \tau^{ij} \) - An integer variable that denotes the total number of destination server nodes in multicast session \( j \) of VDC \( i \). In general, \( 0 \leq \tau^{ij} \leq L_{ij} - 1 \).
- \( \lambda^{sd}_{mn} \) - A real variable that denotes the amount of traffic for a unicast session between node pair \((s, d)\) in VDC \( i \) that traverses physical link \((m, n)\).
- \( \omega^{mn} \) - A binary variable that takes the value of 1 if physical link \((m, n)\) is used; 0, otherwise.
- \( q_s, q_e, q_o \) - Three integer variables that denote the numbers of servers, electronic switch ports, and optical transponders used, respectively.

Our objective is to minimize the total system cost and the total link capacity used when all the VDC requests are embedded.

Objective: minimize

\[
P_s \cdot q_s + P_e \cdot q_e + P_o \cdot q_o + \alpha \cdot (\sum_{i \in \Omega, j \in Y_i} \lambda_{ij}^{sd}) + \sum_{m \in P} \sum_{n \notin NE_m} \sum_{d \in S \cap \Theta_{ij}} \delta_{mn}^{sd} \leq \sum_{t \in T} \sum_{i \in \Omega, j \in Y_i} \lambda_{ij}^{l_r} \sum_{n \in P} \sum_{v \in \Theta_{ij}} \delta^{l_r}_{vm} \]

The embedding of all the VDC requests is subject to the following constraints.

Constraints:
- VM node mapping constraints

\[
\sum_{i \in \Omega, j \in Y_i} \beta_{it} \cdot \delta^{l_r}_{vm} \cdot \beta_{it} \cdot RD^{l_r}_{v} \leq C_r \forall t \in T, i \in \Omega
\]

\[
\sum_{m \in P} \delta^{l_r}_{vm} = 1 \forall i \in \Omega, v \in N_i
\]

\[
\tau^{ij}_{n} \geq \delta_{vm}^{l_r} - \delta^{l_r}_{vn} \forall i \in \Omega, j \in Y_i, n \in P, v \in \Theta_{ij}, v \neq n
\]

\[
P_s, P_e, P_o \]

Normalized costs of a server, an electronic switch port (which is calculated as the total switch cost divided by the total number of switch ports), and an optical transponder, respectively. The cost of a switch port does not include the cost of an optical transponder that is plugged into the switch port.

\[
\alpha
\]

A factor used to weigh two optimization objectives.
Multicast link mapping constraints

\[ \sum_{m \in \mathcal{E}_n} \mu_{jm}^x = \sigma_{ij} \quad \forall i \in \Omega, j \in Y_{ij}, n \in S \]  

(8)

Commodity flow constraints for creating a tree

\[ \sum_{m \in \mathcal{E}_n} \varphi_{ij}^x = \sum_{m \in \mathcal{E}_n} \varphi_{ij}^x \quad \forall i \in \Omega, j \in Y_{ij}, n \in S \]  

(16)

\[ \varphi_{ijn} \leq \left( L_{ij} - 1 \right) \cdot \delta_{ij}^{y_{ij}} \quad \forall i \in \Omega, j \in Y_{ij}, n \in \mathcal{P}, m \in \mathcal{N}_n \]  

(17)

Volume node mapping constraints: Constraint (2) ensures the resource requirement, including CPU, memory, and disk space, of each server at any time. Constraint (3) ensures that each VM node is mapped onto only one physical server node. This is always true since a VM cannot be split and mapped onto different servers. Constraints (4)-(6) decide whether server node \( n \) is a destination server node of session \( j \) in VDC \( i \). Specifically, if server node \( n \) hosts at least one destination VM node of session \( j \) in VDC \( i \), but does not host the source VM node of the session, then \( n \) is considered a destination server node of session \( j \) in VDC \( i \). Constraint (7) finds the total number of destination server nodes in session \( j \) of VDC \( i \).

Multicast link mapping constraints: Constraint (8) ensures that every switch node that belongs to a multicast tree has one incoming edge on the tree. Constraints (9) and (10) ensure that if there is any VM node of session \( j \) in VDC \( i \) mapped on server node \( m \), then the server node must belong to the multicast tree of the session. Constraint (11) ensures that each destination server node must have an incoming edge on the tree of the multicast session. Constraint (12) ensures that each switch node used by a multicast session has at least one outgoing edge on the multicast tree. Constraint (13) ensures that if a server node \( n \) hosts the source VM node of a multicast session and there is at least one destination VM node hosted by a different server node, then server node \( n \) has one outgoing edge on the tree. Constraint (14) ensures that a destination server node of a session does not have any outgoing edge on the tree. Constraint (15) ensures that if there is no VM node mapped onto a server node, then this server node does not have any outgoing edge on the tree.

Commodity flow constraints for creating a tree: Constraints (16)-(23) are the flow-conservation equations for successfully creating a tree topology [39]. Specifically, Constraint (16) ensures that at an intermediate switch node, the total number of outgoing flows equals that of incoming flows. Constraints (17)-(19) ensure that the total number of
outgoing flows from a source server node equals the total number of destination server nodes in a session, and the number of outgoing flows from a destination server node is zero. Constraint (20) ensures that there is no incoming flow for the source server node of a session. Constraint (21) ensures that each destination server node has one incoming flow. Constraints (22) and (23) ensure that if the number of commodity flows in a multicast session that uses a physical link is greater than zero, then this physical link must be used by this multicast session.

The set of “Multicast link mapping constraints” (i.e., (8)-(15)) plus the set of “Commodity flow constraints for creating a tree” (i.e., (16)-(23)) jointly ensure to create a multicast tree for each multicast session.

Unicast link mapping constraints: Constraints (24) and (25) jointly ensure flow conservation for all the unicast sessions.

Link capacity constraint: Constraint (26) ensures that the total capacity of all the unicast and multicast communication sessions of all the active VDCs that traverse a common physical link does not exceed the total capacity of the link.

Active server and switch port constraints: Constraint (27) ensures that if there is a VM node mapped onto a server node, then this server node must be active. Constraints (28)-(31) ensure that if there is any incoming or outgoing traffic for any server node, then this server node must be active. Constraints (32)-(36) decide whether a physical link is used. Constraints (37)-(38) find the total numbers of servers and switch ports used.

The computational complexity of an ILP model is decided by the dominant number of variables and constraints. In this model, the dominant number of variables is at the level of $O(|\Omega| \cdot |V|^2 \cdot |N| \cdot |NE|)$ due to variable $\lambda_{mn}$, where $|\Omega|$ is the total number of VDC requests, $|V|$ is the total number of VMs in each VM node set $N_i$ ($i \in \Omega$), $|N|$ is the total number of physical nodes, and $|NE|$ is the total number of neighbor nodes in each set $NE_m$ ($m \in P \cup S$). The dominant number of constraints is at the level of $O(|\Omega| \cdot |V|^2 \cdot |P| \cdot |NE|)$ due to constraints (35) and (36), where $|P|$ is the total number of server nodes in the physical network.

To minimize the total numbers of server nodes and link capacity used, VM nodes in a VDC should be mapped onto as few server nodes as possible. This, however, would affect the reliability of this VDC as the failure of a single server node would then affect many VM nodes of the VDC [9]. Therefore, to ensure the VDC’s reliability, we add a new constraint (39) to require a server node to hold only up to K VM nodes of a VDC. Without losing generality, this study has set $K=1$ for high reliability.

$$\sum_{m \in NE} \delta_{in} \leq K \quad \forall i \in \Omega, n \in P$$

(39)

In addition, if a unicast session is seen as a special case of a multicast session with only two VM nodes interconnected by one virtual link, then for simplicity we may use a unified model to remove all the unicast parameter $\lambda_{mn}$, variable $\lambda_{mn}^{str}$ and constraints (24)-(25), (30)-(31), and (35)-(36). Also, constraints (26) and (34) can be simplified by removing the unicast variables and parameters, and the objective function can be simplified to

$$P_s \cdot q_s + P_e \cdot q_e + P_0 \cdot q_o + \alpha \cdot \sum_{m \in S \cup \Omega, n \in \Omega, i \in \Omega, j \in \Omega} \lambda_{ij} \cdot \mu_{mn}$$

(40)

5 HEURISTIC ALGORITHM FOR MIXED VDC EMBEDDING

Although the MILP model can provide an optimal solution, the time complexity increases rapidly when the problem scales up, and the model becomes computationally infeasible to use. Because of this, we also develop a heuristic algorithm, which can solve the mixed VDC embedding problem in a short time and can achieve a performance close to that of the MILP model. This heuristic algorithm is described next.

Specifically, as in [3], we divide the server nodes into groups of different sizes. Servers hosted in the same rack are grouped to form a basic group. Then these basic groups are further combined to form larger groups. As shown in Fig. 1, these larger groups can be the combination of any two basic groups or may even consist of all the basic groups. We sort these server groups according to their sizes in ascending order. If there are groups with the same number of servers, then we further sort these groups, in descending order, according to the numbers of active servers in each. The purpose of the second sorting is to enable the group that has more active servers to embed VDCs earlier so as to reduce the use of servers and link capacity. We denote this ordered group list as $L$. For any VDC request, the algorithm scans $L$ to find the first eligible group to embed the VDC. The pseudocode of this VDC embedding algorithm is shown in Fig. 3. The algorithm includes two key steps, i.e., VM node mapping and virtual link mapping, whose details are given next.

![Fig. 3. Pseudocode of the proposed VDC embedding algorithm.](image-url)

**Procedure of VM node mapping:** This step implements the function `MatchNodes()` in the pseudocode of Fig. 3. For each server group that is eligible to embed a VDC, we assign resources available on server nodes in the group to each VM node of the VDC in a way as shown in Fig. 4. We...
first sort the VM nodes in the VDC according to their CPU resource requirements in descending order. We also sort the server nodes in the group according to their remaining CPU capacities in ascending order. In Fig. 4, the VM nodes are displayed in descending order on the left-hand side, and the server nodes are displayed in ascending order on the right side. Then we get a VM node from the left-hand side list to find an eligible server node that can provide sufficient resources. These include CPU, memory, and disk space required by the VM, as well as the ingress and egress capacities of the VM. The egress capacity required by a VM is the sum of traffic demands of all the sessions where this VM is a source node. The ingress capacity is the sum of traffic demands of all sessions where this VM is a destination node. If it is possible for this server node to provide all these capacities, then we use it to map this VM node. We repeat the same process for the subsequent VM nodes in the VM node list until they are all mapped. The right-most column in Fig. 4 shows the resource status of the server nodes after mapping. This mapping process aims to minimize the mismatch of CPU resources between VM nodes and their mapped server nodes, thereby reducing the number of active server nodes required. In the above process, we have also considered the constraint that VM nodes belonging to a common VDC should not be mapped onto a common server node.

**Procedure of virtual link mapping:** This step implements the function BuildMulticastTree() in the pseudocode of Fig. 3. After successfully mapping the VM nodes onto the server nodes, this step maps each communication session by building a multicast tree on a physical tree topology. Here we consider each unicast request as two multicast sessions with two VM nodes and that they are in opposite directions. Before building the multicast tree, we remove all the links from the physical topology that do not have enough remaining capacity to serve the session. For each session, we first create an empty tree \( \mathcal{P} \). Then, we add its source server node and the ToR switch that the server node directly connects to, and the link connecting these two nodes, to \( \mathcal{P} \). Next, for each destination server node, we run Dijkstra’s algorithm based on the physical topology to find a switch node \( S \) in \( \mathcal{P} \) that is the nearest to this server node. Then we find the shortest route \( \mathcal{R} \) from \( S \) to the server node. Finally, we augment \( \mathcal{P} \) by adding the links and nodes on the route \( \mathcal{R} \) that are new to \( \mathcal{P} \). We repeat this process for all the destination nodes and create a multicast tree for the multicast session.

As done for the server node mapping, when mapping virtual links, we also first use as many as possible of those switch nodes that are already active to avoid turning on new switches. New switches are turned on only if the currently active switches cannot accommodate a session.

The main parts of the above algorithm are the procedures of VM node mapping and virtual link mapping, which are of polynomial computational complexities. The computational complexity of the VM node mapping procedure is \( O(|V| \cdot log|V| + |G| \cdot log|G|) \) when fast sorting algorithms are used, where \(|V|\) is the total number of VM nodes in the VDC request and \(|G|\) is the total number of servers in a server group considered. The computational complexity of building a multicast tree is \( O(|N|^2 \cdot |\Theta|) \), where \(|N|\) is the total number of physical nodes and \( |\Theta| \) is the total number of VM nodes in a multicast session. Thus, the computational complexity of the virtual link mapping procedure is \( O(|N|^2 \cdot |\Theta| \cdot |Y|) \), where \(|Y|\) is the total number of sessions in the VDC request. Consequently, the overall computational complexity for embedding a VDC is \( O((|V| \cdot log|V| + |G| \cdot log|G| + |N|^2 \cdot |\Theta| \cdot |Y|) \cdot |L|) \), where \(|L|\) is the total number of server groups considered.

The algorithm described above embeds only one VDC. When embedding multiple VDCs, there are some differences for the algorithms under the static and dynamic VDC request scenarios.

### 5.1 Static Scenario

Under the static scenario, the information on all the VDC requests, including their arriving and holding times, will be given in advance. We aim to minimize the total system cost and the power consumption subject to the condition that all the VDC requests are successfully embedded. For this, we first assume that there are sufficient DC resources including servers and link capacity. Once all the VDCs are embedded, we remove all the inactive equipment (including servers and switches) not actually being used from the system and then find the corresponding performance results in terms of system cost, power consumption, etc.

It is important to observe that the embedding results are influenced by the order, in which VDCs are embedded. Therefore, we consider applying a shuffling process to optimize results. Specifically, we have a time sequence \( T \), and for each time \( t \in T \), there can be multiple VDC requests that are active at \( t \) (in other words, their service durations \( t \)). We first shuffle the times in \( T \) many times, and then for each \( t \in T \), we further shuffle many times the VDC requests that are active, but have not been embedded yet at the time \( t \) being considered. Next, we embed these shuffled unserved VDC requests one by one for each time \( t \) in the shuffled time sequence \( T \). The above embedding process under different shuffling sequences will lead to different system performance in terms of system cost and energy consumption. We calculate the objective value based on (1) for each of the shuffled sequences to select the one with the smallest value of the objective function.
5.2 Dynamic Scenario
Under the dynamic scenario, VDC requests arrive randomly and sequentially. They provide cloud services to users for a certain period and finally depart. Each time a VDC request arrives, we embed it using the above heuristic algorithm. If the embedding process is successful, we allocate resources to this VDC; otherwise, we reject this VDC request. When the lifespan of a VDC is over, we withdraw the resources allocated to it. We keep on simulating this VDC dynamic arrival and departure process. After simulating a certain large number (e.g., 10^6) of VDC requests, we count the total number of rejected VDC requests and calculate the rejection ratio as the number of rejected VDC requests divided by the total number of arriving VDC requests.

6 Simulations and Performance Analyses
In this section, we evaluate the performance of the proposed mixed VDC embedding. We consider both the static and dynamic demand scenarios. Under the static scenario, we consider system cost, link capacity utilization, and power consumption as the evaluation metrics. Under the dynamic scenario, we consider the rejection ratio of VDC requests as an evaluation measure.

6.1 Test Conditions
We employ the VL2 topology of [40] for the DCN. One test case (i.e., Case 1) consists of 15 servers connected by 3 ToR switches, and further connected by 2 aggregation switches and 2 core switches. Another test case (i.e., Case 2) consists of 300 servers, 6 ToR switches, 4 aggregation switches, and 4 core switches. The link between each server and a ToR switch is a twisted copper wire with a 1-Gb/s capacity. The links between the switches are optical fibers, each of which has a 10-Gb/s capacity. Each server hosts 8 CPUs, 64-GB memory, and 500-GB disk space. The price of a 1-Gb/s copper switch port is normalized to be 1 unit. The price of each 10-Gb/s switch port with an optical transponder plug-in is set to be 100 units and the price of a server is set to be 1300 units. Note that all these prices are set based on their current market prices.

In Case 1, the number of VM nodes per VDC (i.e., M) is set to vary from 2 to 8. The number of CPUs required by each VM node varies from 1 to 3, the memory requirement varies from 0 to 32 GB, and the disk space requirement varies from 0 to 100 GB. Multicast sessions in a VDC are randomly generated and the number of sessions per VDC varies from 1 to M. The number of VM nodes in each multicast session varies from 3 to M. As a special case, the number of VM nodes in each unicast session is always 2. The virtual link capacity in each session varies from 40 Mb/s to 100 Mb/s.

In Case 2, we set the number of VM nodes per VDC to vary from 2 to 15, while all the other parameters are the same as those of Case 1. Since a DC may not be able to serve a VDC with a unicast approach when the number of VMs in a multicast session is too large, we limit the number of VMs in a session to 8 or less for the static scenario, so that all the VDCs can be successfully embedded for both unicast and multicast approaches. Of course, such a constraint is not needed for the dynamic scenario.

For performance comparison, we also evaluate the performance based on unicast embedding, where for each destination node in a multicast session, an independent unicast session is set up. Moreover, as the source node needs to process multiple unicast sessions independently for each destination node, additional CPUs are required compared to the implementation based on multicast sessions. Here, we assume that the data delivery process of each session (either multicast or unicast) consumes 1/3 unit of CPU resource at the session source node. Therefore, the number of extra CPUs required when applying the unicast embedding approach to embed a multicast session is [(L_{ij} - 2)/3] where [.] is the ceiling of a decimal number “,” and L_{ij} is the total number of VM nodes in a session (see Section 4.2). In addition, the embedding algorithm employed in the SecondNet framework [3] is also considered for comparison. Specifically, for each VDC, the algorithm first selects a candidate server group, and then builds a bipartite graph with the VM nodes of the VDC on one side and the server nodes in the server group on the other side. Then the algorithm further adds edges between each pair of VM node and server node as long as the server node has enough capacity to hold the VM node. Then, the algorithm transforms the VM mapping problem to a minimum cost flow problem, and completes the VM mapping process by solving the problem. For more details on the algorithm, please refer to [3]. In addition, for the fairness of performance comparison, we also apply the same shuffling process as described earlier to the algorithm of SecondNet when embedding a set of VDC requests under the static scenario.

We employ the commercial software AMPL/Gurobi to solve the MILP model, for which α in (1) is set it to be 0.01 by considering the minimization of the system cost as the first priority, and the MIPGAP is set to be 0.01%. We used Java to implement the heuristic algorithm.

6.2 Simulation Results
6.2.1 Static Scenario
For the static scenario, we consider both Cases 1 and 2. We randomly generate floating-point numbers with a uniform distribution to represent the arriving time and duration of each VDC request. Without losing generality, we normalize the arriving time of each VDC to vary from 0 to 1 and its duration to vary from 0.5 to 1. We use the arrival times of all VDCs to construct the time sequence T (as defined in Section 4.2).

For the shuffling process, the total number of shuffles is dependent on the number of VDCs to be embedded. Specifically, for the smaller case, i.e., Case 1, if the factorial of the number of VDC requests is no more than 10,000, then the number of shuffles for the time sequence is set to be the previous factorial value; otherwise, it is set to be 10,000. Similarly, for shuffling of VDC requests at each time t ∈ T, if the factorial of the number of VDC requests not yet embedded is 10,000 or less, then the number of corresponding shuffles is set to be the previous factorial value; otherwise, it is set to be 10,000. In Case 2, due to its much larger system.
size, we set both the shuffle times for the time sequence and the VDC requests not embedded at each time $t \in T$ to be 100.

We evaluate the performance in terms of system cost, link capacity utilization, and energy consumption, which are analyzed as follows.

1) **System cost**: We first evaluate the total system cost required for accommodating all the VDC requests. The system cost is the sum of the costs of the servers and the electronic and optical switch ports, weighted by their normalized market prices. Fig. 5(a) shows the results of the small test case (i.e., Case 1), in which “Uni” corresponds to the case of using unicast technique to support multicast sessions, “Multi” corresponds to the case of mixed VDC embedding, “Model” corresponds to the results obtained by the MILP model, and “Alg” corresponds to the results obtained by the heuristic algorithm. “SecondNet” corresponds to the results found using the algorithm of SecondNet [3].

![System Cost vs # of VDCs](image)

Fig. 5. System costs of the different approaches.

In addition, comparing the results of the MILP model and the heuristic algorithm, we see that the results of the heuristic algorithm match those of the MILP model reasonably well. They have almost the same performance except when the numbers of VDCs are around 7–9. Actually, the difference here is minor as it is only the cost difference of one server and one electronic switch port, i.e., for these VDC values, the heuristic algorithm requires one server and one electronic switch port more than the MILP model.

Similar studies were also carried out for Case 2, for which results are shown in Fig. 5(b). Here we do not provide the results for the MILP model as it is computationally intractable to run the model for a large number of servers and VDCs. However, we provide the results obtained by the SecondNet embedding approach as a benchmark. Comparing the results of the unicast and mixed embedding, we again see that the latter significantly outperforms the former, i.e., by up to 19%. This once again shows the efficiency of the proposed mixed embedding approach. Moreover, comparing with the benchmark scheme, we find that the mixed embedding approach significantly outperforms the SecondNet embedding approach, i.e., by up to 20%. This, therefore, shows the efficiency of the proposed mixed VDC embedding approach. In addition, it is interesting to see that the unicast embedding approach proposed in this paper and the embedding approach of SecondNet perform very similarly. This is because both of them use multiple unicast virtual links to construct a multicast tree, but do not employ the multicasting capability of the physical tree as proposed in the mixed VDC embedding scheme.

2) **Link capacity utilization**: As the second optimization objective, we also minimize the link capacity used. Fig. 6(a) shows the results of this for Case 1. Overall, we see that performance trends are similar to those observed for the system cost. The mixed embedding approach can significantly outperform the unicast embedding approaches, i.e., by more than 45%. Moreover, the results of the MILP model and the heuristic algorithm based on mixed embedding are very close to each other, which indicates the efficiency of the proposed algorithm. A similar performance comparison is also made for the larger test case, i.e., Case 2, whose results are shown in Fig. 6(b). As before, we see that the mixed embedding approach is more efficient and significantly outperforms the unicast embedding approach, i.e., by up to 53%. SecondNet’s performance is almost the same as that of the unicast embedding approach as they are both based on unicast link mapping.

![Link Capacity Utilization vs # of VDCs](image)
3) **Power consumption:** We also evaluate the power consumption of the system based on the term $E_1 \cdot q_s + E_2 \cdot q_0$ where $E_1$, $E_2$, and $E_0$ are the average power consumptions of a server, a 1-Gb/s electrical switch port, and a 10-Gb/s optical switch port, respectively. We set their values to be 400 W, 9 W, and 82 W, respectively. Note that these data were once again derived from the power consumption data of actual products in the market. Fig. 7 shows the results of power consumption for the different approaches. The mixed embedding approach always consumes less power (i.e., by up to 24%) than that for the unicast embedding approach. The result trend is very similar to that of the system cost shown in Fig. 5. This is reasonable as the multicast embedding approach can reduce the numbers of servers and switch ports compared to the unicast embedding approach, thereby reducing power consumption.

**6.2.2 Dynamic Scenario**

For the dynamic scenario, we only consider Case 2. We evaluate the performance of the heuristic algorithm of mixed embedding in term of the VDC rejection ratio, which is defined as the total number of rejected VDC requests divided by the total number of VDC requests that arrived during the simulation run.

As described earlier, under the dynamic scenario, VDC requests arrive randomly and sequentially. The arrival and holding time of each VDC request is not known in advance. In our experiment, we assume that VDC requests arrive following a Poisson arrival process with arrival rate $\lambda$ requests/hour and the duration of each VDC service follows an exponential distribution with mean $1/\mu$. Here we normalize the mean duration $1/\mu$ to be 1 hour. A total of $10^6$ VDC request arrivals were simulated to compute the VDC rejection ratio.

The results of the VDC rejection ratio with an increasing arrival rate are shown in Fig. 8. We can see that the mixed embedding approach can significantly outperform both the unicast embedding approach and the approach employed by SecondNet. This is reasonable because the mixed embedding approach needs fewer physical resources to provision a VDC, which enables provisioning of more VDCs given the limited physical resources of the system. Consequently, the mixed embedding approach can achieve a lower VDC rejection ratio.
We also evaluate how the capacity required by each VDC multicast session can impact the VDC rejection ratio. Fig. 9 illustrates the VDC rejection ratio changes with an increasing average capacity requirement of each multicast session. Here the arrival rate of VDC requests is set to be 100 requests/hour, which corresponds to a 100-erlang traffic load. The average capacity of each multicast session changes from 50 Mb/s to 200 Mb/s with a 50-Mb/s step-size. For each average capacity X Mb/s, we then randomly generate a specific capacity whose value is uniformly distributed within the range of [X-30, X+30] Mb/s for each multicast session. Based on the results in Fig. 9, we can see that with an increasing average capacity of multicast sessions, the VDC rejection ratio increases since each multicast session consumes more resources at a higher average capacity requirement. Moreover, we observe that the performance difference between the unicast embedding and multicast embedding approaches becomes more significant under a higher average capacity requirement. The improvement of the rejection ratio is only 0.022 when the average capacity requirement of a multicast session is 50 Mb/s, while this improvement increases to 0.45 when the average capacity requirement of a multicast session grows to 250 Mb/s. This is because, compared to the mixed embedding approach, the unicast embedding approach wastes more system resources at a higher average capacity requirement when it uses multiple unicast virtual links to construct a multicast tree for a multicast session.

6.2.3 Computational Time

As in other publications on VDC embedding [1], [3], [22], we also evaluate the time consumption for embedding a VDC when its size scales up. We consider Case 2 because it considers a larger-size problem, and here the test conditions are the same as those presented earlier in Section 6.1. We also limit the number of VMs in each session to 8 for each VDC as was done for the static scenario earlier. We measure the execution times by embedding different VDC requests on a 3.40-GHz Intel Core i7 with 8-GB memory and. The results of computation times are shown in Fig. 10. From the results, we can see that the proposed mixed embedding approach is significantly faster than the approach used by SecondNet. Moreover, as the scale of a VDC grows, the difference between them becomes even more pronounced. Specifically, when the size of a VDC is 28, the approach used by SecondNet takes 580% more time than that of the mixed embedding approach. This is reasonable because when mapping VMs, the approach of SecondNet needs to build a bipartite graph and then run minimum cost flow algorithm based on the graph, whose computational complexity is higher than that of the proposed approach in this paper. In addition, because the unicast and multicast embedding approaches adopt the same category of algorithm, they have very close time consumptions for embedding a VDC.

7 Conclusion

Considering the inefficiency of using the unicast technique to transmit data for multicast services, we propose a mixed VDC embedding approach, which employs the multicasting capability of switches to carry multicast sessions in VDCs. To minimize the system cost and the resource consumption, we develop a MILP model and a heuristic algorithm to solve the problem of proposed mixed VDC embedding. Simulation results show that for the static VDC demand, the proposed mixed embedding approach can significantly outperform its unicast counterpart to reduce the system cost, link capacity unitization, and power consumption by up to as much as 37%, 55%, and 38%, respectively. In addition, the heuristic algorithm is efficient and performs close to the MILP model. We also evaluate the performance of the proposed mixed VDC embedding approach in a dynamic VDC demand scenario. It is found that the mixed VDC embedding approach is also efficient to significantly reduce the VDC rejection ratio compared to the unicast-only approaches.

Acknowledgment

This work was jointly supported by National Natural Science Foundation of China (NSFC) (61671313), the Science and Technology Achievement Transformation Project of Jiangsu Province, PR China (BA2016123), and by a grant
from the Research Grants Council of the Hong Kong Special Administrative Region, China (CityU 11200318).

REFERENCES


Chao Guo is currently pursuing his master’s degree with the School of Electronic and Information Engineering, Soochow University. His research interest is network virtualization.

Kai Xu is now with Department of Electronic Engineering, City University of Hong Kong, Kowloon, Hong Kong as a Ph.D. student. His research interests are in the areas of broadband networks, optical networks, and network virtualization.

Sanjay Kumar Bose (SM’91) received the B.Tech. Degree from IIT Kanpur in 1976, and the Master’s, and the Ph.D. from SUNY Stony Brook, USA, in 1977 and 1980, respectively. He was with the Corporate R&D Centre of the General Electric Co., Schenectady, NY, USA, till 1982. He joined IIT Kanpur as an Assistant Professor and became a Professor in 1991. He left IIT Kanpur in 2003 and joined the School of EEE, NTU, Singapore, as a Faculty Member. In 2008, he left NTU to join IIT Guwahati where he is currently a Professor with the Department of EEE. He has been involved in various areas in the field of computer networks and queueing systems and has published extensively in the area of optical networks and network routing. He is a fellow of IETE (India), and a member of Sigma Xi and Eta Kappa Nu.

Moshe Zukerman (M’87–SM’91–F’07) received his B.Sc. and M.Sc. degrees from the Technion, Israel, and his Ph.D. degree in engineering from University of California, Los Angeles, in 1985. He was an independent Consultant with the IRI Corporation and a Postdoctoral Fellow with the University of California, Los Angeles, in 1985–1986. During 1986–1997, he was with Telstra Research Laboratories. He also taught and supervised graduate students at Monash University in 1990–2001. During 1997–2008, he was with The University of Melbourne, Victoria, Australia. In 2008 he joined City University of Hong Kong as a Chair Professor of Information Engineering, and a team leader. He has over 300 publications in scientific journals and conference proceedings. He has served on various editorial boards and contributed to numerous conference technical committees.