CoMan: Managing Bandwidth Across Computing Frameworks in Multiplexed Datacenters

Wenxin Li, Deke Guo, Alex X. Liu, Keqiu Li, Heng Qi, Song Guo, Ali Munir, and Xiaoyi Tao

Abstract—Inefficient bandwidth sharing in a datacenter network, between different application frameworks, e.g., MapReduce and Spark, can lead to *inelastic* and *skewed* usage of link bandwidth and *increased completion times* for the applications. Existing work, however, either solely focuses on managing computation and storage resources or controlling only sending/receiving rate at hosts. In this paper, we present CoMan, a solution that provides global in-network bandwidth management in multiplexed data centers, with two goals: improving bandwidth utilization and reducing application completion time. CoMan first designs a novel abstraction of virtual link groups (VLGs) to establish a shared bandwidth resource pool. Based on this pool, CoMan implements a three-level bandwidth allocation model, which enables elastic bandwidth sharing among computing frameworks as well as guarantees network performance for the applications. CoMan further improves the bandwidth utilization by devising a VLG dependency graph and solves an optimization problem to guide the path selection using a $\frac{3}{2}$-approximation algorithm. We conduct comprehensive trace-driven simulations as well as small-scale testbed experiments to evaluate the performance of CoMan. Extensive simulation results show that CoMan improves the bandwidth utilization and speeds up the application completion time by up to $2.83\times$ and $6.68\times$, respectively, compared to the ECMP+ElasticSwitch solution. Our implementation also verifies that CoMan can realistically speed up the application completion times by $2.32\times$ on average.

Index Terms—Data-parallel computing frameworks, multiplexed datacenter, bandwidth management.

1 Introduction

Datacenters are increasingly hosting a mixed variety of applications: from Internet services to data-parallel, HPC and scientific applications. Driven by this wide range of applications, researchers and engineers have been developing a diverse set of computing frameworks such as MapReduce [1], Spark [2], and Pregel [3]. Moreover, new computing frameworks will continue to emerge, and no single framework will be optimal for all applications. Therefore, organizations will want to run multiple frameworks *in the same datacenter*, so as to choose the best framework for each application, and at the same time reduce the investments of datacenters [4, 5].

Unfortunately, simply *multiplexing* a datacenter among different computing frameworks can lead to several negative consequences. As we know, many applications of these computing frameworks involve massive amounts of data transfers in datacenter networks. These data transfers can account for more than 50% of the application completion time [6, 7]. Since each framework lacks network optimization to speed up such data transfers, the *application completion time* is usually extended. Meanwhile, the distribution of these data transfers can be non-uniform on datacenter links. This leads to the *skew use of link bandwidth*. That is, some links are overloaded while some others possibly experience extremely low bandwidth utilization. In the context of multiple frameworks coexisting in the same datacenter, the extended application completion time and the skew problem of network utilization will be more serious. As a consequence, the scalability of deployed applications can be significantly restricted. Moreover, because these frameworks are developed independently, there is no way to perform the collaborative bandwidth sharing across frameworks, thus creating the *inelastic use* of the bandwidth. In other words, the bandwidth allocated to one framework cannot be used by others frameworks even if it is unused. Therefore, to improve application completion times, it is important to design an intelligent bandwidth management mechanism to avoid inelastic usage of the link bandwidth and a path selection mechanism to avoid the skewed usage of network resources in a datacenter shared by multiple computing frameworks.

The existing work on network bandwidth management either considers only a single application framework in the network or does not consider the network resources while sharing network bandwidth across frameworks. First, some existing mechanisms only focus on managing the CPU and memory resources across these network-bound frameworks, and ignore the network resource utilization [5, 8, 9]. Second, some mechanisms only consider transport layer solutions for bandwidth sharing by adapting the sending/receiving rates of the flows to minimize flow or coflow completion times [6, 10, 11]. However, these mechanisms do not consider the collaborative use of in-network bandwidth across multiple computing frameworks. Lastly, some existing network sharing mechanisms [12–14] consider band-

Corresponding authors: Heng Qi; Alex X. Liu; Keqiu Li.

- *W. Li, K. Li, H. Qi and X. Tao are with the School of Computer Science and Technology, Dalian University of Technology, No 2, Linggong Road, Dalian 116023, China. Email: liwenxin@mail.dlut.edu.cn, {keqiu, hengqi}@dlut.edu.cn, taoxiaoyi@mail.dlut.edu.cn.*
- *D. Guo is with the College of Information System and Management, National University of Defense Technology, Changsha 410073, P.R. China. E-mail: guodeke@gmail.com.*
- *A. X. Liu and A. Munir are with the Department of Computer Science, Michigan State University, East Lansing, MI 48824-1226 USA. Email: alexliu@cse.msu.edu, munirali@msu.edu.*
- *S. Guo is with the Department of Computing, The Hong Kong Polytecnic, Hung Hom, Kowloon, Hong Kong SAR. Email: song.guo@polyu.edu.hk.*

width management of multiple VMs in datacenter networks but do not solve skewness issue (or resolve congestion) as they do not consider routing or path level isolation.

In this paper, we propose CoMan, a bandwidth management and network path selection mechanism for shared datacenter networks. CoMan aims to improve the network bandwidth utilization and reduce the application completion times using following three design principles: *1) allow the elastic use of link bandwidth among multiple computing frameworks*, *2) provide performance guarantees to applications*, and *3) avoid the skewed usage of network bandwidth*.

To distribute network bandwidth among competing frameworks and do path selection to improve application performance, it is crucial to build a shared resource pool. However, it is extremely difficult to build such pool for in-network bandwidth of a datacenter because the bandwidth resources on the datacenter links are tightly coupled, as a flow's progress depends on the transfer rates it gets on all the links along the connection path. To tackle this challenge, CoMan devises a novel abstraction of virtual link groups (VLGs), where each VLG is a group of decoupled links. All the links in a datacenter are encapsulated into different VLGs, and a large shared bandwidth resource pool is established accordingly. Based on this pool, CoMan further performs the path selection to avoid the skew use of link bandwidth. To this end, we devise a VLG dependency graph and formulate a path selection problem to minimize the residual network bandwidth. Specifically, we present a $\frac{3}{2}$-approximation algorithm to solve the NP-hard path selection problem.

CoMan implements a three-level bandwidth allocation model to manage inter and intra application bandwidth. The first level reserves the bandwidth for different frameworks, in a max-min fashion, to enable the elastic bandwidth sharing among computing frameworks. The second level performs the inter-application bandwidth allocation with respect to the guaranteed network performance of its applications. To efficiently manage the network bandwidth among different frameworks, we use three algorithms, i.e., the weighted fair sharing, bandwidth preemption, and bandwidth borrowing. The third level performs intra-application bandwidth allocation for the individual flows, which determines the application performance [10].

The first challenge for CoMan is how to design virtual link groups (VLGs). To address this challenge, we leverage the datacenter architecture while encapsulating the links into different VLGs. The commodity data centers typically follow the design of three-level tree (Core, Aggregation, and Edge) [15], and network flows follow fixed paths from the source to the destination. We therefore characterize the links into either upstream or downstream links, and group the ingress or egress links of the switch to form a VLG.

The second challenge in CoMan design is how to make bandwidth sharing elastic. To address this challenge, we propose a preemptive bandwidth sharing model, where a framework can share its spare bandwidth with other frameworks and later preempt it upon increase in its own load. We propose a bandwidth preemption and bandwidth borrowing models to solve this problem.

The third challenge in CoMan design is how to avoid link utilization skewness. To address this challenge, we propose a path selection mechanism that leverages the VLG abstraction. The bandwidth manager in CoMan allocates bandwidth on a shared pool of links and, during path selection, the flow can choose path with the lower load, from its shared pool, to transfer data. This helps distribute flows evenly across the links and avoid link utilization skewness.

To evaluate the performance of CoMan, we conduct both large-scale trace-driven simulations and small-scale testbed implementation. The comprehensive simulation results show that applications complete up to $4.55\times$ and $6.68\times$ faster on average, in comparison to the widely used ECMP+Per-Flow and ECMP+Elasticswitch solutions [12, 16, 17], respectively. In addition, compared to ECMP+Per-Flow and ECMP+Elasticswitch, CoMan improves the bandwidth utilization by up to $2.88\times$ and $2.83\times$, respectively. Furthermore, our implementation verifies that CoMan can realistically speed up the application completion time by up to $2.32\times$ on average.

In summary, the main contributions of this paper include:

- We address the challenging problem of managing the in-network bandwidth across multiple computing frameworks in a multiplexed datacenter. Specifically, we focus on two goals: improving the bandwidth utilization and reducing the application completion time.
- We propose CoMan, a collaborative bandwidth management architecture, which contains three key components: bandwidth virtualization model, three-level bandwidth allocation model, and the VLG dependency graph-based path selection.
- We conduct extensive trace-driven simulations as well as a small-scale testbed implementation to evaluate the performance of CoMan.

The remainder of this paper is organized as follows. Section 2 summarizes the related work. Section 3 presents the motivation and challenges for the problem of managing the in-network bandwidth across multiple computing frameworks in a single datacenter. Section 4 presents the overview of CoMan and the novel bandwidth virtualization model. In Section 5, we discuss the three-level bandwidth allocation model and Section 6 presents a dependency graph-based path selection algorithm. In Section 7, we evaluate and analyze the performance of CoMan. We discuss current limitations of CoMan and revelent future research in Section 8, and we conclude in Section 9.

2 Related Work

Existing works on network bandwidth management and datacenter scheduling can be divided in three categories such as network sharing, network scheduling, in datacenter networks, and the resource management in cluster frameworks.

2.1 Network sharing in datacenter networks

Existing network bandwidth sharing mechanisms for datacenter networks do not address the skewness and elastic usage of network bandwidth simultaneously. For example,

Faircloud [18] proposed a set of desirable properties and allocation methods for achieving the VM-pair level fairness when sharing the bandwidth on congested links. Guo et al. [19, 20] proposed a game theory based bandwidth allocation strategy, which guarantees minimum bandwidth, and at the same time, achieves VM-pair level fairness. Liu et al. [21] proposed a novel distributed rate allocation algorithm based on the Logistic Model under the control-theoretic framework, and leverage the feedback of link utilization from switches to control the rate of senders. Since the total bandwidth guarantee is less than the physical bandwidth in the solution of [21], they further propose SoftBW that can provide efficient bandwidth/fairness guarantee with bandwidth over commitment [22]. Seawall [13] allocates bandwidth to sources based on the established hypervisor-to-hypervisor tunnels among physical servers, to achieve the per-source fair sharing on congested links. NetShare [23] leverages weighted fair queues for proportional bandwidth sharing among different tenants on congested links. ElasticSwitch [12] utilizes the spare bandwidth from unreserved capacity, to achieve both the minimum bandwidth guarantees and work conservation properties. Chen et al. [24] focus on the performance-centric fairness for sharing datacenter network bandwidth among data-parallel applications, and present distributed algorithms to derive such fairness sharing. These network sharing methods mainly concern on allocating bandwidth among competing entities, i.e., address elastic usage but do not address the skewness problem as they do not consider routing while making bandwidth sharing decisions.

2.2 Network scheduling in datacenter networks

The network scheduling based mechanisms share the network among flows of a single application or multiple applications by carefully designing the orderings of flows to be transported in the network. These mechanism can be categorized into two folds, the flow-level scheduling and the coflow-level scheduling. For the flow-level schedulers, some of them focus on traffic management with the aim of increasing network throughput or achieving load balancing, e.g., Hedera [11] and MicroTE [25], and others design transport-level mechanism to minimize the flow completion time (e.g., pFabric [26], PASE [27], L^{2}DCT [28] and PIAS [29]). AC/DC TCP [14] is a virtual congestion control protocol that allows the network administrators to control per VM congestion control in a multi-tenant network. For the coflow scheduling, which accounts for the collective behaviors of flows, Chowdhury et al. [6] first propose a centralized architecture to share bandwidth at both intra-coflow and inter-coflow level. Baraat [30] integrates both the advantages of FIFO and FS strategies for inter-coflow scheduling. Varys [10] further improves coflow scheduling by combining priority-based inter-coflow scheduling and weighted-based intra-coflow scheduling. Aalo [31] focuses on scheduling coflows without requiring a prior knowledge of coflows. These coflow schedulers donot account for the bandwidth management as most of them only focus on adjusting the sending rate at the hosts. No matter scheduling at flow-level or coflow-level, the network scheduling methods do not consider the flexible bandwidth sharing across multiple computing frameworks.

2.3 Resource management and scheduling in cluster computing frameworks

Existing cluster management and scheduling frameworks ignore the network resource while making sharing decisions and only focus on sharing the compute and storage of the computing frameworks. For example, the fair scheduler in Hadoop [32] and delay scheduler in Dyrad [33] try to achieve data locality, by considering compute resources and input data storage location, to avoid network transfers as much as possible. For the resource management in cluster computing, Omega [8] mainly focuses on designing a distributed, multi-level job scheduling method. Similarly, Mesos [5] uses an offer-based approach to design a two-level scheduler, while Yarn [9] leverages a request-based approach to implement a two-level scheduler. However, these resource managers in cluster computing framework are only applicable for sharing the computation and storage resources among diverse computing frameworks. They leave the network resource sharing to the underlying transport mechanisms, which make the applications of computing frameworks to suffer from unpredicted network performance and to experience arbitrary completion times. Moreover, these resource managers cannot simply be extended to the network management due to the lack of network-level abstraction. Recently, NEAT [34] proposed a network state aware task placement framework, however, it does not address the skewed usage of network resources as it does not consider path selection while making task placement decisions.

3 Motivation and Challenges

Deploying multiple computing frameworks in a single datacenter can significantly slow down the completion of applications belonging to these frameworks. For a better intuition of this point, we run two applications on a same datacenter: one is MapReduce application and the other one is Spark application. Both of them implement a `WordCount` function. The datacenter consists of 8 servers and 10 4-port switches, and every two servers connect to an edge switch (detailed topology of this datacenter will be introduced in Section 7.2). For each edge switch, we let one connecting server run MapReduce application and the other one run Spark application. As such, each application occupies 4 homogenous servers, implying that they can get equal amount of CPU and memory resources. We first run these two applications in turn, and then run them simultaneously. This implies that each application is first executed in an exclusive mode and then is executed in a shared mode. We calculate the extended completion time of an application as its completion time in the shared mode minus that in the exclusive mode. Fig. 1 plots the extended completion time for these two applications, under various sizes of input data. We find that coexisting frameworks in a datacenter considerably extends application completion time. This phenomenon can be more serious as the input data size increases, for both MapReduce and Spark applications. Under the same configurations of CPU and memory resources, the root cause for such extended completion time is the lacking of an efficient in-network bandwidth managing scheme.

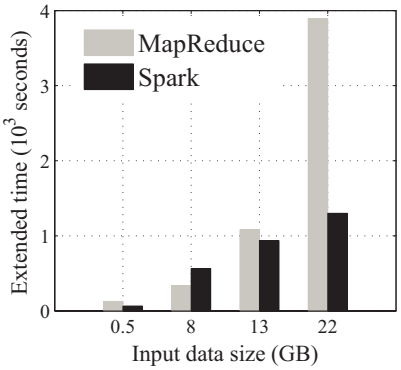


Fig. 1. The extended completion time when deploying multiple computing frameworks in a data center.

TABLE 1
Example settings for each computing framework (CF).

Bandwidth Settings (Mbps)	CFs		
	1	2	3
Minimum guaranteed bandwidth for each application	10	20	25
Maximum guaranteed bandwidth for each application	15	25	30
Minimum guaranteed bandwidth for each framework	20	30	40
Maximum guaranteed bandwidth for each framework	30	40	45

To efficiently manage the in-network bandwidth across multiple computing frameworks, we argue that both bandwidth allocation and routing should be considered.

Potential benefit of three-level bandwidth allocation: When multiple computing frameworks coexist in a datacenter, the bandwidth should be allocated at three-level rather than one-level or two-level. One-level bandwidth allocation refers to distributing bandwidth at the level of individual flows, e.g., per-flow sharing scheme [17], which, however, cannot account for collective behaviors of flows. This is because that the application completion time depends on the time it takes to complete all the flows, instead of the time to complete the individual flows. The two-level bandwidth allocation, distributing bandwidth at the application-level, e.g., [16, 31], can grasp the application-level semantics. Unfortunately, it may cause the applications of a certain computing framework to exclusively occupy the network resources, leaving no bandwidth for applications of other frameworks. The is mainly because that the two-level bandwidth allocation makes no effort to limit the bandwidth that each computing framework can use. The three-level bandwidth allocation means that the bandwidth is first allocated to each computing framework, then to each application, and finally to each flow. In such a case, the application-level semantics can be accurately grasped, yet the bandwidth can have a chance to be elastically shared among different computing frameworks.

We use an example to illustrate how the three-level bandwidth allocation can enable such elastic bandwidth sharing. Consider there are a 100Mbps link and three computing frameworks. Each framework as well as each application have been configured with a minimum and a maximum guaranteed bandwidth, as shown in Table 1. When CF1 has no applications, its minimum guaranteed bandwidth can then be borrowed by CF2 and CF3, such that the applications of CF2 and CF3 can all be guaranteed with a minimum bandwidth. Once CF1 increases its load, i.e., there are two new applications, CF1 needs to preempt the bandwidth that

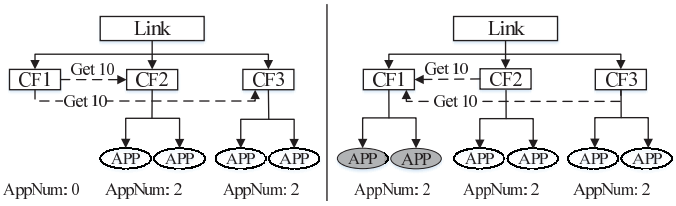


Fig. 2. An illustrative example of how the three-level bandwidth allocation can support elastic bandwidth sharing among multiple computing frameworks.

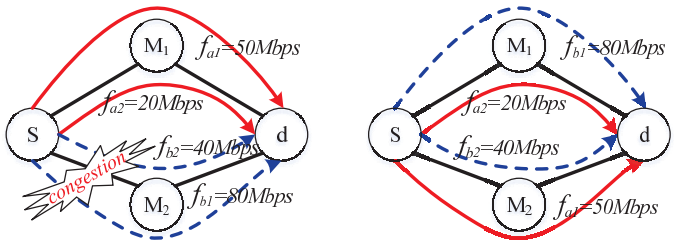


Fig. 3. A motivating example of load-balancing routing.

it previously lent to CF2 and CF3. In such a case, the two new applications of CF1 can then be guaranteed with a minimum bandwidth.

Potential benefit of load-balancing routing: It has been revealed that the input data of applications in computing frameworks like MapReduce and Spark are not necessarily uniform, and often exhibit significant skew [35–37]. Such skewed distribution of the input data can make a small number of links to be utilized significantly more than others, leading to *the skew use of link bandwidth*. Such skewness may help to save energy in datacenter networks [38–40], as long as the flow bandwidth demands on each path is under the link capacity. However, it can increase the possibility of congestion and failure [41, 42], and thus reduces fault tolerance of the application. Hence, a load-balancing routing scheme is desired. Load balancing routing ensures that no link is overloaded and thus improves the overall system performance. The main aim of load-balancing routing is to assign flows to appropriate links and balance the bandwidth usage among all links. Efficient load-balancing routing helps in high link bandwidth utilization and helps in implementing failover, enabling scalability, avoiding bottlenecks and reducing application completion time.

For a better intuition of this point, we use an example to show the benefit of load-balancing routing. In this example, there are two applications: Application a has flows f_{a1} and f_{a2} with the bandwidth demands of 50Mbps and 20Mbps respectively; Application b has flows f_{b1} and f_{b2} with the bandwidth demands of 80Mbps and 40Mbps respectively. The link bandwidth are all 100Mbps. Fig.3(a) shows a case of randomized routing by equal-cost multipath (ECMP). Under ECMP scheme, the flows of application b are congested on path $s \rightarrow M_2 \rightarrow d$, failing to guarantee the flow bandwidth demands and affecting the application completion time. However, when we applying a load-balancing routing scheme for flows of both applications a and b, all flows can successfully be routed with bandwidth guarantee, yet the link bandwidth utilization can significantly be improved, as shown in Fig.3(b).

Fundamental challenges: The above examples look straightforward with simple settings. But the general prob-

lem of jointly considering bandwidth sharing and routing to manage the in-network bandwidth across multiple computing frameworks in a datacenter can be difficult, due to the following challenges. *First*, a flow's progress depends on the bandwidth allocation on all links along its routing path, yet an application's performance depends on all its flows' progress, leading to tight coupling among the datacenter links, network flows and applications. *Second*, the arrival pattern of applications belonging to each computing framework is unknown in advance, yet is difficult to be accurately predicted. In most practical scenarios, we can only get the information of applications that have arrived. So, how can we enable efficient elastic bandwidth sharing among multiple computing frameworks in such highly dynamic environment? *Third*, the bandwidth allocation strategy will directly impact the routing decision for each flow because each link has a fixed amount of bandwidth capacity. This implies that bandwidth allocation and routing are deeply intertwined with each other.

4 CoMan: Collaborative Bandwidth Management

4.1 CoMan Overview

In this work, we propose CoMan, a bandwidth sharing and routing framework to share network resources among competing frameworks, in a multiplexed datacenter. To manage the in-network bandwidth, CoMan first reserves an amount of bandwidth for each computing framework which then distributes the reserved bandwidth between its applications and flows. We propose a novel abstraction layer for network bandwidth management to address the elasticity and skewness problems. To manage routing, CoMan further computes the routing paths for the individual flows of each application, before the application is admitted to the datacenter network. The key idea is to achieve global coordination among multiple frameworks, with the aim of improving the bandwidth utilization and reducing application completion times. To this end, CoMan focuses on the following three design principles:

- **Elastic bandwidth usage:** The reserved bandwidth for each computing framework should be either used or lent out, such that the bandwidth efficiency can be improved in a flexible way.
- **Guaranteed network performance:** Each application should be assigned minimum bandwidth to meet its performance guarantees.
- **Bandwidth skewness avoidance:** The path selection mechanism for the individual flows should distribute flows evenly in the network to avoid the skewed link bandwidth utilization.

CoMan proposes a novel abstraction layer to do bandwidth sharing and network path selection. The CoMan architecture is shown in Fig. 4. At the lowest level, CoMan decouples the network resources from the underlying datacenter links using the proposed novel abstraction of virtual link groups (VLGs). Each VLG encapsulates a set of decoupled links, that carries the network traffic. By taking advantage of the VLG abstraction, CoMan builds a VLG-based datacenter network and maintains a large

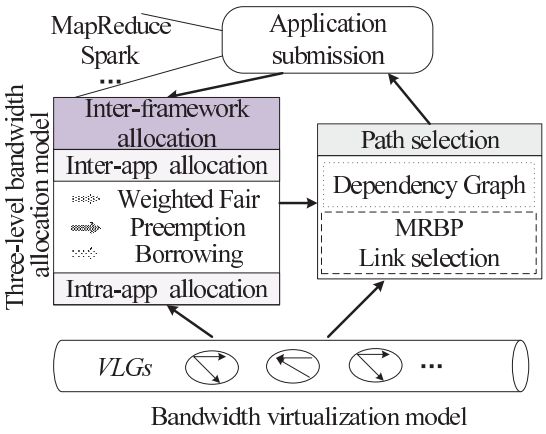


Fig. 4. The overview of our collaborative bandwidth management architecture — CoMan.

shared bandwidth resource pool. Given the shared resource pool, CoMan implements a three-level bandwidth allocation model to distribute the bandwidth between multiple computing frameworks, multiple applications and among the flows of the same application. To compute the paths, to meet the bandwidth guarantees and evenly distribute load in the network, CoMan constructs a VLG dependency graph, where each VLG is scheduled to perform the link selection with the aim of scrabbling all flows' routing paths. The link selection problem is formulated as a minimum residual bandwidth problem (MRBP) and is solved through a $\frac{3}{2}$ approximation algorithm.

4.2 Network Resource Virtualization Model

The key building block of CoMan is the VLG abstraction. Below, we first present the abstraction overview and then discuss how it can be used to build a VLG-based datacenter network.

4.2.1 Abstraction of virtual link groups (VLGs)

The bandwidth resources on all intra-datacenter links can be consolidated into a large shared resource pool to do bandwidth management and path selection. A shared resource pool is becoming increasingly important in datacenter environments to improve the resource utilization and reduce the network management costs [43]. For example, the shared CPU or memory resource pool, encapsulating resources on hundreds or even thousands of servers through the virtualization technique, has been widely applied in multi-user cloud environments [44]. However, simply encapsulating the link bandwidth into a shared pool can be problematic because of ***link coupling*** problem. ***link coupling*** happens due to the close ties among the network flows and the datacenter links because a flow utilizes the bandwidth on all the links along its routing path. To tackle this challenge and decouple the network resources from the underlying links, we propose a novel virtual link group (VLG) abstraction that consolidates the bandwidth resources on all intra-datacenter links into a large shared resource pool. VLG can be formally defined as follows:

Definition 1. *A **virtual link group (VLG)** is defined as a group of decoupled links, in which the flows can be transferred on any link.*

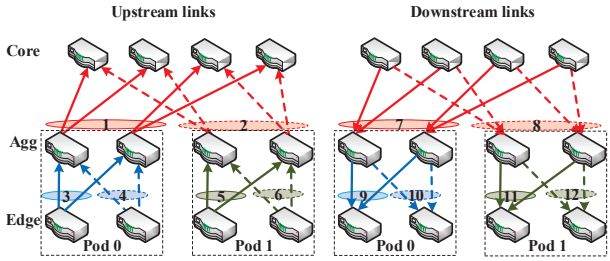


Fig. 5. Encapsulating links in a partial 4-radix Fat-Tree topology.

We leverage datacenter architecture while encapsulating the links into different VLGs. The commodity data centers typically follow the design of three-level tree (Core, Aggregation, and Edge) [15, 45], and when designing tree-like data centers [46], most network flows follow fixed paths, going up from the source first and going down to the destination late. We therefore characterize the links into either upstream or downstream links, and consider two link encapsulation polices:

- ***Upstream link encapsulation:*** For a given pod, the upstream links sharing a *same head node (edge switch)* can be encapsulated into a VLG. Similarly, the upstream links on egress of *a same pod* can also be encapsulated into a VLG because flows may come out from any one aggregation switch.
- ***Downstream link encapsulation:*** For a given pod, the downstream links sharing *a same tail node (edge switch)* can be encapsulated into a VLG. Similarly, the downstream links directing to *a same pod* can be encapsulated into a VLG.

Insights: We highlight the following benefits that such VLG abstraction can bring to bandwidth resource allocation and traffic engineering in datacenter networks. First, such VLG abstraction can significantly reduce the network management costs, and can provide flexible design choice for bandwidth allocation strategy. This is because that the VLG can abstract entire datacenter network resources into a large shared bandwidth resource pool. Second, the VLG group can significantly simplify the routing optimization for network flows. As we will show in Section 6, by constructing the VLG dependency graph, once a flow has determined to be routed on a link in a certain VLG a, the link for composing this flow's routing path can directly be identified in the VLG that depends on VLG a.

4.2.2 VLG-based datacenter network

Fig. 5 depicts an example of the link encapsulation in a partial 4-radix Fat-Tree network, which contains 12 VLGs in total. Given the link encapsulation polices, we can generate multiple VLGs in a datacenter, with the collection of links belonging to each VLG being disjoint. To this end, we build a virtual bandwidth resource pool which contains bandwidth on multiple VLGs. Note that we do not consider the edge links for encapsulation, as we do not control task placement and leave that to the task schedulers.

Under the VLG abstraction, we focus on a VLG-based datacenter network model, with key notations in Table 2. We model the datacenter network as a directed graph $\mathcal{G}=(\mathcal{N},\mathcal{E})$, where $\mathcal{N}=\{v_1,v_2,\cdots,v_N\}$ represents the set of all VLGs and $\mathcal{E}$ is the set of edges that connect these

TABLE 2
Key Parameters in VLG-based Datacenter Network.

Notations	Definitions
$\mathcal{N}$	The set of VLGs
$\mathcal{K}$	The set of computing frameworks
$\mathcal{M}$	The set of applications
$\mathcal{L}_i$	The set of links encapsulated in VLG $v_i \in \mathcal{N}$
C_i	The bandwidth capacity of v_i
$\mathcal{F}(v_i)$	The set of flows hosted by v_i
$\mathcal{F}(v_i,a_j)$	The set of flows hosted by VLG v_i and belonged to application $a_j \in \mathcal{M}$
$\mathcal{M}(v_i)$	The set of applications hosted by v_i
$\mathcal{M}(v_i,t_k)$	The set of applications hosted by v_i and belonged to computing framework $t_k \in \mathcal{K}$
$R_{i,k}^{min}$	The minimum guaranteed bandwidth configured for t_k in VLG v_i
$R_{i,k}^{min-}$	The amount of consumed bandwidth in $R_{i,k}^{min}$
$R_{i,k}^{min+}$	The amount of available bandwidth in $R_{i,k}^{min}$
$R_{i,k}^{max}$	The maximum reservable bandwidth configured for t_k in VLG v_i
$\alpha_{i,k}$	The minimum guaranteed bandwidth per application configured for t_k in VLG v_i
$\beta_{i,k}$	The maximum consumable bandwidth per application configured for t_k on VLG v_i
$\Phi_{i,k}$	The amount of bandwidth to be preempted for t_k in v_i
$\Psi_{i,k}$	The amount of bandwidth to be borrowed for t_k in v_i
$r_{i,k}^{j}$	The amount of bandwidth allocated to $a_j \in \mathcal{M}(v_i,t_k)$
$x_{i,j}^{f}$	The amount of bandwidth allocated to flow $f \in \mathcal{F}(v_i,a_j)$
μ_i^f	The amount of bandwidth allocated to flow $f \in \mathcal{F}(v_i)$

VLGs. Specifically, an edge connecting two VLGs represents a dependency relationship (which will be introduced in Section 6). Let $\mathcal{L}_i$ denote the set of links in VLG v_i, and $C_i=\sum_{l\in\mathcal{L}_i} c_l$ denote the bandwidth capacity of v_i, where c_l is the bandwidth capacity of link $l\in\mathcal{L}_i$. Let $\mathcal{K}=\{t_1,t_2,\cdots,t_K\}$ denote the set of coexisting *frameworks* in the datacenter, with t_k denoting the k-th framework. Consider that there are a set of applications submitted by all frameworks in a given time period, $\mathcal{M}=\{a_1,a_2,\cdots,a_M\}$, which carry a set $\mathcal{F}$ of flows. Let a_j index the j-th application, and denote $\mathcal{F}(a_j)$ as the set of flows in $a_j\in\mathcal{M}$.

We divide the individual flows among VLGs, and accordingly divide the applications among those VLGs, based on the distribution of flows in each application. Based on the VLG abstraction, we observe that a flow typically consumes bandwidth on either 2 VLGs or 4 VLGs, determined by whether it's an intra-pod flow or an inter-pod flow. For ease of presentation, let $\mathcal{F}(v_i)$ denote the set of flows hosted by v_i, with $\mathcal{F}(v_i,a_j)$ being the set of application a_j's flows that are hosted by v_i. In addition, let $\mathcal{M}(v_i)$ denote the set of applications hosted by v_i, with $\mathcal{M}(v_i,t_k)$ being the set of applications belonging to t_k. It should be noted that an application may be simultaneously distributed to multiple VLGs, as its flows may traverse multiple links that belong to multiple VLGs. Therefore, $\mathcal{M}(v_i)\cap\mathcal{M}(v_{i'})\neq\varnothing$ may be feasible for some $v_i\neq v_{i'}$, while $\mathcal{M}(v_i,t_k)\cap\mathcal{M}(v_i,t_{k'})=\varnothing$ is always feasible.

5 CoMan Bandwidth Allocation Mechanism

CoMan implements a three-level bandwidth management mechanism for the inter-framework, inter-application, and intra-application bandwidth sharing, which we discuss below.

5.1 Inter-framework bandwidth allocation

To share the bandwidth among frameworks in an *elastic* manner, CoMan allocates the bandwidth to VLGs in a max-min manner, so as to avoid the exclusive use of bandwidth

by any computing framework. Data centers are increasingly hosting a mixed variety of computing frameworks, and are typically multi-user environments, where hundreds of applications may run simultaneously [32]. In such a case, the bandwidth is shared among multiple computing frameworks in an *elastic* way by first providing them required bandwidth and then sharing the remaining bandwidth in fair share manner among the frameworks.

To achieve max-min bandwidth sharing, CoMan first reserves a minimum guaranteed bandwidth for each computing framework t_k on each VLG v_i. If the applications of a certain computing framework require more bandwidth than the minimum reserved bandwidth, then more bandwidth can be reserved. Otherwise, the idle part of the minimum reserved bandwidth can be lent to applications of other frameworks. To support elastic bandwidth sharing, the applications can later preempt the bandwidth borrowed by other computing frameworks, so as to obtain at least its minimum bandwidth share. We can define it formally as follows: let $R_{i,k}^{min}$ and $R_{i,k}^{max}$ denote the minimum guaranteed bandwidth and maximum reservable bandwidth for t_k in v_i, respectively. The *max-min bandwidth constraint model* for inter-framework bandwidth reservation becomes:

- Each t_k has the minimum guaranteed $R_{i,k}^{min}$ and the maximum reservable bandwidth $R_{i,k}^{max}$ on each v_i, where $R_{i,k}^{min} \leq R_{i,k}^{max}, \forall v_i \in \mathcal{N}, \forall t_k \in \mathcal{K}$.
- The sum of the maximum reservable bandwidth is allowed to exceed the VLG's bandwidth capacity; that is, $\sum_{t_k \in \mathcal{K}} R_{i,k}^{max} \geq C_i, \forall v_i \in \mathcal{N}$.
- To avoid congestion, the sum of the minimum guaranteed bandwidth should not exceed the VLG's bandwidth capacity; i.e., $\sum_{t_k \in \mathcal{K}} R_{i,k}^{min} \leq C_i, \forall v_i \in \mathcal{N}$.

Given this model, datacenter operators can flexibly set parameters of $R_{i,k}^{min}$ and $R_{i,k}^{max}$ for each framework. For example, if the scientific applications dominate the datacenter and require more resources, then the minimum and maximum bandwidth for the corresponding framework are set to be higher than other frameworks.

5.2 Inter-application bandwidth allocation

The next step in bandwidth allocation is to share the bandwidth reserved for each framework among the applications. To provide bandwidth guarantee and performance isolation for applications, we assign at least the minimum required bandwidth of each application and formulate a min-max problem, since it provides a lower bound on the application performance irrespective of the communication patterns of other applications [18, 19]. It can be formulated as: let $\alpha_{i,k}$ denote the minimum guaranteed bandwidth per application configured for framework t_k on VLG v_i. In addition, let $\beta_{i,k}$ denote the maximum consumable bandwidth per application configured for t_k and v_i. Where, $\alpha_{i,k} \leq \beta_{i,k}$.

We formulate inter-application bandwidth sharing as a **max-min bandwidth allocation (MMBA)** problem. Let $r_{i,k}^j$ denote the amount of bandwidth allocated to the application $a_j \in \mathcal{M}(v_i, t_k)$, and define $\Omega_{i,k} \triangleq [\alpha_{i,k}, \beta_{i,k}]$. CoMan allocates inter-application bandwidth allocation independently across different VLGs because a VLG may carry flows from a variety of applications and all the VLGs along a flow path may not have same amount of bandwidth available. This implies that the allocated bandwidth of an application on each of the involved VLGs is not necessarily the same. Therefore, for each VLG v_i, we have the following **MMBA** problem:

Algorithm 1 Bandwidth preemption for t_k in v_i

```
Initialize pd=0 and compute Φ_{i,k} (Eq. (5));
while pd≤Φ_{i,k} do
    Search a t_{k'}∈K with nonzero λ^{k'}_{i,k}>0;
    pd+= min{λ^{k'}_{i,k}, Φ_{i,k}−pd};
    λ^{k'}_{i,k}−= min{λ^{k'}_{i,k}, Φ_{i,k}−pd};
if R^{min+}_{i,k}+pd ≥ α_{i,k}M_{i,k} then
    Invoke the weighted fair sharing algorithm;
else
    Ensure r^j_{i,k}←α_{i,k} for as many applications as possible;
```

$$\max \sum_{t_k \in \mathcal{K}} \sum_{a_j \in \mathcal{M}(v_i, t_k)} 1_{r_{i,k}^j > 0} \quad (1)$$

$$\textbf{Subject to:} \sum_{a_j \in \mathcal{M}(v_i, t_k)} r_{i,k}^j \leq R_{i,k}^{max}, \forall t_k \in \mathcal{K}, \quad (2)$$

$$\sum_{t_k \in \mathcal{K}} \sum_{a_j \in \mathcal{M}(v_i, t_k)} r_{i,k}^j \leq C_i, \quad (3)$$

$$\textbf{Variable:} r_{i,k}^j \in \Omega_{i,k} \vee \{0\}, \forall t_k \in \mathcal{K}, \forall a_j \in \mathcal{M}(v_i, t_k), \quad (4)$$

where $1_{r_{i,k}^j > 0}$ is an indicator variable that is 1 if $r_{i,k}^j > 0$ and 0 otherwise. The objective of admission control in Eq. (1) is to maximize the number of accepted applications. As enforced by Eq. (2), the consumed bandwidth of each t_k cannot exceed the maximum reservable bandwidth $R_{i,k}^{max}$. Eq. (3) is a capacity constraint for VLG v_i. Eq. (4) ensures that the decision variable is either 0 or between the range of $[\alpha_{i,k}, \beta_{i,k}]$. Note that, one can obtain different objectives such as minimizing the application completion time, given the above constraints. In this paper, we mainly focus on maximizing the number of accepted applications for improving the scalability of deployed applications.

Next, we combine three bandwidth allocation algorithms to derive an optimal solution to the **MMBA** problem. Let the available and consumed bandwidth in the minimum guaranteed bandwidth ($R_{i,k}^{min}$) be denoted by $R_{i,k}^{min+}$ and $R_{i,k}^{min-}$, respectively, where, $R_{i,k}^{min+} + R_{i,k}^{min-} = R_{i,k}^{min}$ ($\forall v_i \in \mathcal{N}, \forall t_k \in \mathcal{K}$).

5.2.1 *Weighted fair sharing*

The *weighted fair sharing algorithm* is used, for each framework t_k in v_i, if the available minimum bandwidth $R_{i,k}^{min+}$ is sufficient to guarantee the minimum bandwidth $\alpha_{i,k}$ for all applications in $\mathcal{M}(v_i, t_k)$. It first tries to allocate $\alpha_{i,k}$ amount of bandwidth to each application. Next, it divides the remaining bandwidth between each application a_j based on its weight w_j, under the constraint of maximum consumable bandwidth per application ($\beta_{i,k}$). For simplicity, the weight w_j is defined as the number of flows in the application a_j, to achieve fairness among applications at the network scale.

Algorithm 2 Bandwidth borrowing for t_k in v_i

1: Initialize tb=0 and compute $\Psi_{i,k}$ based on Eq. (6);
2: Update tb=$\min\{Z_i, \Psi_{i,k}\}$ and Z_i−=$\min\{Z_i, \Psi_{i,k}\}$ if Z_i>0;
3: **while** $tb \leq \Psi_{i,k}$ **do**
4: Search a $t_{k'} \in \mathcal{K}$ with $R_{i,k'}^{min+} > \alpha_{i,k} M_{i,k'}$;
5: tb+=$\min\{R_{i,k'}^{min+} - M_{i,k'}\alpha_{i,k}, \Psi_{i,k}\}$;
6: $\lambda_{i,k'}^{k}$+=$\min\{R_{i,k'}^{min+} - M_{i,k'}\alpha_{i,k}, \Psi_{i,k}\}$;
7: Allocate $\alpha_{i,k}$ to the un-served applications in $\mathcal{M}(v_i, t_k)$ until the borrowed bandwidth tb is used up.

5.2.2 *Bandwidth preemption*

The *bandwidth preemption* algorithm, as shown in **Algorithm** 1, is invoked when the available minimum bandwidth $R_{i,k}^{min+}$ is insufficient to accommodate all the applications demands in $\mathcal{M}(v_i, t_k)$ and some bandwidth has been borrowed by other computing frameworks. Formally, for each v_i, let $\lambda_{i,k}^{k'}$ represent the amount of minimum bandwidth that computing framework t_k previously lent to $t_{k'}$. Let $M_{i,k}$ denote the number of applications in $\mathcal{M}(v_i, t_k)$. Then, the amount of bandwidth that needs to be preempted for t_k in v_i can be calculated as follows:

$$\Phi_{i,k} = \min\{M_{i,k}\beta_{i,k} - R_{i,k}^{min+}, \sum_{t_{k'} \in \mathcal{K}} \lambda_{i,k}^{k'}\}. \quad (5)$$

Algorithm 1 begins by searching a $t_{k'}$ with nonzero $\lambda_{i,k}^{k'}$ until $\Phi_{i,k}$ amount of bandwidth is located for preemption (Steps 2-5). It then allocates the bandwidth based on the summation of $\Phi_{i,k}$ and $R_{i,k}^{min+}$. Similarly, if each application can be guaranteed with a minimum bandwidth $\alpha_{i,k}$, it invokes the *weighted fair sharing* algorithm (Step 7). Otherwise, it tries to allocate $\alpha_{i,k}$ amount of bandwidth to as many applications as possible (Step 9).

5.2.3 *Bandwidth borrowing*

The *bandwidth borrowing* algorithm, as shown in **Algorithm** 2, is invoked when there is spare bandwidth in the network and some applications have more requirement to achieve their performance. Note that applications in an overloaded framework can only obtain a minimum guaranteed bandwidth. We therefore calculate the amount of bandwidth to be borrowed for t_k in v_i as follows:

$$\Psi_{i,k} = \min\{R_{i,k}^{max} - R_{i,k}^{min}, M_{i,k}\alpha_{i,k} - R_{i,k}^{min+} - \Phi_{i,k}\}. \quad (6)$$

Let Z_i=C_i−$\sum_{t_k \in \mathcal{K}} R_{i,k}^{min}$ denote the amount of bandwidth in VLG v_i, which is shareable among all computing frameworks. To borrow $\Psi_{i,k}$ amount of bandwidth for t_k, **Algorithm** 2 first seeks the available shared bandwidth (Step 2). It then borrows the available minimum bandwidth from $t_{k'} \in \mathcal{K}$ until $\Psi_{i,k}$ is filled up (Steps 3-6). Finally, it allocates $\alpha_{i,k}$ amount of bandwidth to those un-served applications until the borrowed bandwidth is used up (Step 7).

After the processing of these three algorithms, we finally update the available bandwidth $R_{i,k}^{min+}$ and consumed bandwidth $R_{i,k}^{min-}$ for all $v_i \in \mathcal{N}$ and $t_k \in \mathcal{K}$.

Theorem 1. *The above allocating algorithms ensure an optimal solution for the MMBA problem, if $\alpha_{i,k}$ satisfies*

$$\alpha_{i,k} \leq \min\{\frac{R_{i,k}^{min}}{M_{i,k}}, \frac{C_i - \sum_{t_k \in \mathcal{K}} R_{i,k}^{min-}}{M_i}\}, \forall v_i, \forall t_k, \quad (7)$$

where M_i is the number of applications hosted by VLG v_i.

Proof: Clearly, the optimal value for the **MMBA** problem is M_i. Considering three conditions:

- if $\alpha_{i,k}M_{i,k} \leq R_{i,k}^{min+}, \forall v_i, t_k$, only the weighted fair sharing will be invoked. Certainly, each application can obtain a minimum guaranteed bandwidth.
- If for all v_i and t_k, $\alpha_{i,k} \leq (R_{i,k}^{min+} + \Phi_{i,k})/M_{i,k} \leq R_{i,k}^{min}/M_{i,k}$. Then, the bandwidth preemption algorithm will be invoked, and each application can also be guaranteed with a minimum bandwidth.
- Some frameworks may still have applications failed to obtain any bandwidth, after executing the weighted fair sharing and bandwidth borrowing algorithm. In this case, we denote the set of frameworks that hold unserved applications as $\mathcal{K}_1$, and as well denote the set of the remaining frameworks as $\mathcal{K}_2$. Leveraging the fact that the total borrowed bandwidth should not exceed the bandwidth that can be borrowed, we have $\sum_{t_k \in \mathcal{K}_1} M_{i,k}\alpha_{i,k} - R_{i,k}^{min+} - \Phi_{i,k} \leq Z_i + \sum_{t_k \in \mathcal{K}_2}(R_{i,k}^{min+} - M_{i,k}\alpha_{i,k})$. Substituting $Z_i = C_i - \sum_{t_k \in \mathcal{K}} R_{i,k}^{min}$, we yield $M_i\alpha_{i,k} \leq C_i - \sum_{t_k \in \mathcal{K}} R_{i,k}^{min} + \sum_{t_k \in \mathcal{K}_2} R_{i,k}^{min+} + \sum_{t_k \in \mathcal{K}_1} \left(R_{i,k}^{min+} + \Phi_{i,k}\right)$. We can easily check that the minimal value for the right side of the above equation is $C_i - \sum_{t_k \in \mathcal{K}} R_{i,k}^{min-}$. Thus, we have $\alpha_{i,k} \leq (C_i - \sum_{t_k \in \mathcal{K}} R_{i,k}^{min-})/M_i$.

Finally, we can infer from the above three conditions that Theorem 1 is proved. □

5.3 Intra-application bandwidth allocation

Once an application has been allocated bandwidth on each involved VLG, CoMan assigns the allocated bandwidth to individual flows of the application. Note that different computing frameworks may not necessarily use the same intra-application bandwidth allocation strategy, as these frameworks may have different data flow computing model. For completeness, we present a referenced intra-application bandwidth allocation method to derive the flow-level bandwidth allocation.

As $\mathcal{M}(v_i, t_k) \cap \mathcal{M}(v_i, t_{k'}) = \varnothing$ for any $t_k \neq t_{k'}$ in a given v_i, we number all $r_{i,k}^{j}$ ($\forall t_k \in \mathcal{K}, \forall a_j \in \mathcal{M}(v_i, t_k)$) for v_i in order, and drop the index k. Accordingly, we get the bandwidth r_i^j allocated to each $a_j \in \mathcal{M}(v_i)$. For each flow $f \in \mathcal{F}(v_i, a_j)$ of size $d_{j,f}$, the bandwidth $x_{i,j}^{f}$ allocated to it can be calculated as $x_{i,j}^{f} = r_i^j \times d_{j,f} / (\sum_{f \in \mathcal{F}(v_i, a_j)} d_{j,f})$. This implies that the bandwidth allocated to each flow is proportional to the flow data size, and we can easily check that $\sum_{f \in \mathcal{F}(v_i, a_j)} x_{i,j}^{f} = r_i^j$. This method enforces all the flows of a same application to have the equal flow completion times, and accordingly will not make any flow become the bottleneck for poor performance.

Finally, it should be noted that a flow may obtain different amounts of bandwidth in different VLGs. To pursue work-conserving property, CoMan only allocates the minimal bandwidth ($\min_{v_i} x_{i,j}^{f}$) to this flow on the involved VLGs. The remaining bandwidth can be recycled to serve more flows and more applications, by deploying a centralized controller [6].

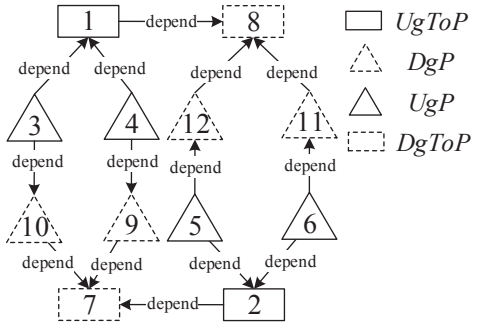


Fig. 6. An example of the VLG dependency graph in the case of a partial 4-radix Fat-Tree datacenter.

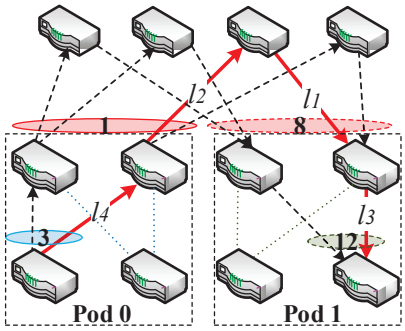


Fig. 7. VLG dependency graph based path selection.

6 VLG Dependency Graph Based Path Selection

In this section, we first present the VLG dependency graph and then present the VLG selection algorithm for the individual flows.

6.1 Virtual link group dependency graph

The bandwidth allocation algorithm reserves the bandwidth across VLGs in the datacenter. Next step is to compute the routing paths for individual flows, so as to quickly transmit flows and solve the *skewed* use of the link bandwidth by carefully planning the routing path for each flow. To this end, we select one link for each flow from the involved VLGs along the path such that the flow should be able to compose a feasible routing path. This implies that the link selection on different VLGs has implicit orderings. More concretely, if two VLGs host a same flow and this flow has been determined to pick a link from the first VLG, then the flow should be placed on a dependent link in the second VLG.

Therefore, based on above insights, we devise a dependency relationship on the abstracted graph $\mathcal{G}=(\mathcal{N},\mathcal{E})$. Specifically, an edge $e_{ii'}\in\mathcal{E}$ connecting VLG v_i to VLG $v_{i'}$ represents that v_i depends on $v_{i'}$. We call the new graph with dependency relationships as the "**VLG dependency graph**", which contains three types of dependency relationships as follows:

- An *upstream VLG within a pod (UgP)* depends on *the upstream VLG on the top of this pod (UgToP)*, as well as *the downstream VLG within this pod (DgP)*.
- An *upstream VLG on the top of a pod (UgTop)* depends on *the downstream VLGs on the top of all pods (DgToP)*, except that pod.
- A *downstream VLG within a pod (DgP)* depends on the *downstream VLG on the top of this pod (DgToP)*.

The VLG dependency essentially means that if VLG a depends on VLG b and a flow selects a certain link in VLG b, then the link in VLG a that can compose the routing path for this flow should be directly identified. In such a case, each flow only picks one link on each involved VLG, and thus the routing path for each flow can be established. Note that if two links share a mutual switch and the VLG with the first link depends on the VLG with the second link, then the first link depends on the second link. Fig. 6 shows an example of such graph in the case of a partial 4-radix Fat-Tree (in Fig. 5). Fig. 7 shows an example of selecting path for a flow, which traverses four VLGs $(3,1,8,12)$, from the *leftmost* edge switch to the *rightmost* edge switch. Since VLG 8 does not depend on any VLG, we start with the link selection on it. If the selected link is the red link l_1, then we observe that link l_2 depends on l_1. Therefore, this flow must utilize the link l_2. Similarly, we can pick links l_3 and l_4 for this flow.

Algorithm 3 ScheduleGraph($\mathcal{G}$)

1: Initialize $isScheduled_i$=0, $\forall v_i\in\mathcal{N}$;
2: **while** $isScheduled \neq 1$ **do**
3: Search a VLG v_i with the in-degree being 0 in $\mathcal{G}$;
4: *LinkSelection(v_i)*;
5: $isScheduled_i$=1;
6: Remove the edges associated with v_i in $\mathcal{G}$;

CoMan uses **Algorithm** 3 on the VLG dependency graph to decide the link selection orderings of VLGs. The algorithm starts by searching a v_i with the in-degree being 0 (Step 3) and then invokes the *LinkSelection(v_i)* algorithm to perform the link selection for flows on the scheduled v_i (Step 4). Finally, it updates the VLG dependency graph with the scheduled VLGs, and deletes the associated edges (Step 6).

6.2 Link selection

The link selection on v_i is to select a link $l\in\mathcal{L}_i$ for each flow $f\in\mathcal{F}(v_i)$, with the aim of improving the link bandwidth utilization. We formulate such a link selection as the **minimum residual bandwidth problem (MRBP)**. Since $x_{i,j}^f$ has been obtained for each $a_j\in\mathcal{M}(v_i)$ and each $f\in\mathcal{F}(v_i,a_j)$ in a given v_i, we actually obtain the bandwidth allocated to each flow in v_i. Specifically, let μ_i^f denote the bandwidth allocated to flow $f\in\mathcal{F}(v_i)$. To formally define **MRBP**, let $I_{f,l}$ be the decision variable, representing whether flow f is placed on link l. Let $c_l^{left}=c_l-\sum_{f\in\mathcal{F}(v_i)}\mu_i^f I_{f,l}$ denote the residual bandwidth on link $l\in\mathcal{L}_i$. The **MRBP** problem is formulated as follows:

$$\min_{I_{f,l}} \sum_{l\in\mathcal{L}_i} c_l^{left} \tag{8}$$

$$\textbf{Subject to:} \sum_{f\in\mathcal{F}(v_i)} \mu_i^f I_{f,l} \le c_l, \forall l\in\mathcal{L}_i, \tag{9}$$

$$\sum_{l\in\mathcal{L}_i} I_{f,l} = 1, \forall f\in\mathcal{F}(v_i), \tag{10}$$

$$\textbf{Variable:}\ I_{f,l}\in\{0,1\}, \forall l\in\mathcal{L}_i, \forall f\in\mathcal{F}(v_i). \tag{11}$$

The objective of MRBP is to minimize the residual bandwidth. Eq. (9) enforces the total consumed bandwidth on a link not to exceed the link bandwidth capacity. Specifically, each flow can only pick one link in $\mathcal{L}_i$, as shown in Eq. (10). This optimization problem is an integer optimization problem, and appears to be in the form of Generalized Assignment Problem (GAP) that is NP-hard [47].

Therefore, we propose a heuristic based solution to solve MRBP problem, as shown in **Algorithm** 4. For each scheduled v_i, we first assign links to flows that have already selected the related dependent links (Step 1). Then, we remove those flows from the set $\mathcal{F}(v_i)$ (Step 2), and update the residual bandwidth for each $l \in \mathcal{L}_i$ (Step 3). By now, there still exist a set of remaining flows, which are then sorted in the increasing order of their allocated bandwidth (Step 4). Finally, for each flow, we search a link with the maximum residual bandwidth in the set $\{l | c_l^{left} \geq \mu_i^f, \forall l \in \mathcal{L}_i\}$, and place this flow on the available link (Steps 5-8). The following theorem proves the approximate ratio of this algorithm.

Theorem 2. *By applying Algorithm 4, the approximation factor for the MRBP problem is, at most,* $\frac{3}{2}$.

Proof: We first transform the *MRBP* problem to an equivalent problem as follows:

$$\max \sum_{l \in \mathcal{L}_i} \sum_{f \in \mathcal{F}(v_i)} \mu_i^f I_{f,l} \text{ s.t. Eqs (9), (10), (11).} \quad (12)$$

Let $\mathcal{OPT}$ denote the optimal value for Eq. (12). It is clear that $\mathcal{OPT} \geq \max_f \mu_i^f$ and $\mathcal{OPT} \geq \frac{1}{|\mathcal{L}_i|} \sum_{f \in \mathcal{F}(v_i)} \mu_i^f$. Consider the link l with maximum load wd_l. Let f be the last flow assigned to the link l. When f was assigned, the link l must have a load smaller than the average load. Then we have

$$wd_l = (wd_l - \mu_i^f) + \mu_i^f \leq \frac{1}{|\mathcal{L}_i|} \sum_{f' \in \{\mathcal{F}(v_i) - \{f\}\}} \mu_i^{f'} + \max_{f \in \mathcal{F}(v_i)} \mu_i^f.$$

This immediately shows that the approximation factor is 2. However, we now prove that such an approximation factor can be $\frac{3}{2}$ if slightly more careful analysis is performed. Note that if there are at most $|\mathcal{L}_i|$ flows in $\mathcal{F}(v_i)$, then optimal solution is to place each flow on a link. If there are more than $|\mathcal{L}_i|$ flows, then there is at least one link in the optimal solution that must get 2 of the first $|\mathcal{L}_i|+1$ flows. Meanwhile, the bandwidth of these flows is at least as big as $\mu_i^{|\mathcal{L}_i|+1}$. Thus, $\mathcal{OPT} \geq 2\mu_i^{|\mathcal{L}_i|+1}$. Similarly, we consider f to be the last flow assigned to the link l with maximum wd_l. Here, we assume that $f > |\mathcal{L}_i|+1$, or else the optimal solution can be generated. As flows are sorted, we have $\mu_i^f \leq \mu_i^{|\mathcal{L}_i|+1}$ and

$$wd_l \leq \frac{1}{|\mathcal{L}_i|} \sum_{f' \in \mathcal{F}(v_i) - \{f\}} \mu_i^{f'} + \mu_i^f \leq \mathcal{OPT} + \frac{\mathcal{OPT}}{2} = \frac{3}{2}\mathcal{OPT}.$$

Thus, Theorem 2 is proved. □

Remarks: One may question that the VLG dependency graph based path selection is still flow level path allocation, and this has already been realized by many existing traffic routing approaches. For example, Hedera [11] focuses on how to distribute flows to balance the traffic load in the network. It should be noted that our path selection is based on the outputs of the three-level bandwidth allocation mechanism, which has already accounted for the collective behaviors of flows belonging to a same application. That is to say, our path selection essentially investigates how to distribute the bandwidth demands of flows belonging to the same application evenly into the network, such that the application-level performance can be guaranteed in real datacenter network.

Algorithm 4 LinkSelection(v_i)

1: Assign $f \in \mathcal{F}(v_i)$ to l if f has been assigned to l's ($l \in \mathcal{L}_i$) dependent links.
2: Remove the flows from $\mathcal{F}(v_i)$, with $\sum_{l \in \mathcal{L}_i} I_{f,l} = 1$;
3: Update $c_l^{left}, \forall l \in \mathcal{L}_i$;
4: Sort all $f \in \mathcal{F}(v_i)$ in the decreasing order of μ_i^f;
5: **for** each $f \in \mathcal{F}(v_i)$ **do**
6: $\quad l \leftarrow \arg\max_{\{l | c_l^{left} \geq \mu_i^f, \forall l \in \mathcal{L}_i\}} c_l^{left}$;
7: $\quad I_{f,l} = 1$;
8: $\quad$ Update $c_l^{left} = c_l^{left} - \mu_i^f$;

7 Performance Evaluation

In this section, we evaluate CoMan using large-scale simulations and a small-scale testbed implementation.

7.1 Large-scale trace-driven simulation

Network topology: We simulate a 16-radix Fat-tree topology, with 1024 servers and 4096 directed links resulting in a total of 2048 upstream and downstream links. The bandwidth capacity of each link is set to 1Gbps, which is the common case in some data centers [48]. Based on the link encapsulating polices, all links in such a 16-radix Fat-tree are partitioned into 288 VLGs.

Datasets: Our experiments are conducted on the daily trace from Google cluster [49]. The trace contains the statistics of application (Job) submissions during a period of 29 days and each application comprises many tasks. The *scheduling class* in the trace indicates the type of the task, and its value ranges from 0 to 3, i.e., 0 refers to the least latency-sensitive tasks, while 3 indicates the most latency-sensitive tasks. In our experiment, we assume that each *scheduling class* in the trace reflects the importance degree of each computing framework, and accordingly this trace covers 4 computing frameworks. Given the trace, we extract the information of applications in an approximate 7-day duration, which contains a 1000-interval period of time with each interval being 10 minutes. The total number of the extracted applications is 151082. The number of applications fall into the four types are 76750, 31905, 28710, 13717, respectively. For each extracted application, we generate flows among its tasks using a many-to-many traffic pattern. Each flow's data size is set to be a random value within $[0, 64]$Mb because the input data size of a task is typically less than 64Mb in those computing frameworks, i.e., MapReduce [1].

Parameter settings: In our simulation, the minimum guaranteed and maximum reservable bandwidth for each computing framework accounts for 20% and 60% of a given VLG's bandwidth capacity, respectively, i.e., $R_{i,k}^{min} = 0.2 \times C_i$, $R_{i,k}^{max} = 0.6 \times C_i$ $\forall v_i \in \mathcal{N}$, $\forall t_k \in \mathcal{K}$. Without loss of generality, the minimum guaranteed bandwidth per application is set to be the same for all computing frameworks and all VLGs, i.e., $\alpha_{i,k} = \alpha, \forall v_i, t_k$. Similarly, the maximum bandwidth per application $\beta_{i,k}$ ($\forall v_i, t_k$) is set to be 1Gbps.

To investigate the impact of the minimum guaranteed bandwidth per application, we record the number of rejected applications under different settings of α. Table 3 indicates that the number of rejected applications grows up as b_1 increases. In this paper, we only present the evaluation results in two scenarios, α=25Mbps and 50Mbps, due to the page limitation. To ease the presentation, let "CoMan (25)" and "CoMan (50)" denote two such scenarios.

TABLE 3
The number of rejected applications

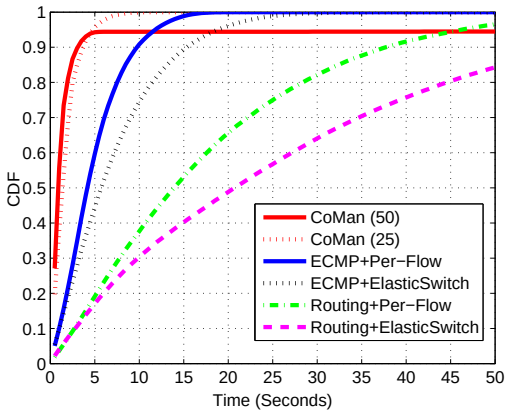
α (Mbps)	25	30	35	40	45	50
Num. of Rejected Apps	0	447	1059	2828	5070	8444

We compare CoMan with the following four schemes.

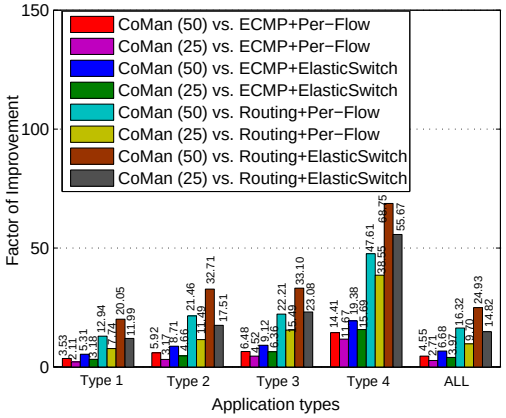
- **ECMP+Per-Flow**: all the flows are routed by ECMP (a randomized routing scheme) and compete fairly for the link bandwidth [16, 17].
- **ECMP+ElasticSwitch**: an integration of the ECMP routing and the ElasticSwitch bandwidth sharing method [12]. ElasticSwitch first partitions the hose-model guarantee of a virtual machine (VM) into VM-to-VM guarantees, and then uses weighted-fair sharing strategy to enforce VM-to-VM rate. The weight of the flows between each VM pair is the corresponding VM-to-VM guarantee. To simulate ElasticSwitch, each server in the Fat-tree network is treated as a VM, and the hose-model guarantee of each VM is set to be 1Gbps.
- **Routing+Per-Flow**: all the flows are routed by a heuristic based load balancing routing scheme and compete fairly for the link bandwidth. This load balancing routing selects path, going up from the source first and going down to the destination late, for each flow. Moreover, when a flow traverses a switch (Core, Aggregation, or Edge) with multiple links, it will choose a link with least workload to route this flow.
- **Routing+ElasticSwitch**: an integration of the heuristic based load balancing routing and the ElasticSwitch bandwidth sharing method.

Performance metric: For comparison, we use the *factor of improvement* as a performance metric. For instance, compared to "ECMP+Per-Flow" or "Routing+ElasticSwitch", the factor of improvement on the application completion time (ACT) is computed as $\frac{\text{ACT in ECMP+Per-Flow}}{\text{ACT in CoMan}}$ or $\frac{\text{ACT in Routing+ElasticSwitch}}{\text{ACT in CoMan}}$. For bandwidth allocation of application (BAA) and the link bandwidth utilization (LBU), such a metric is defined as $\frac{\text{BAA (LBU) in CoMan}}{\text{BAA (LBU) in ECMP+Per-Flow}}$ or $\frac{\text{BAA (LBU) in CoMan}}{\text{BAA (LBU) in Routing+ElasticSwitch}}$.

The application completion time (ACT): Fig. 8(a) first shows the CDF of the application completion time (ACT) for all schemes. We observe that CoMan performs better than all of the schemes: ECMP+Per-Flow, ECMP+ElasticSwitch, Routing+Per-Flow, and Routing+ElasticSwitch. Specifically, the percentage of applications completing within 5s is 94.26% and 95.74% for CoMan (50) and CoMan (25), respectively, compared to 44.43% for ECMP+ElasticSwitch, 60% for the ECMP+Per-Flow, 19.13% for the Routing+Per-Flow, and 17.03% for the Routing+ElasticSwitch. Moreover, all



(a) CDF of application completion time



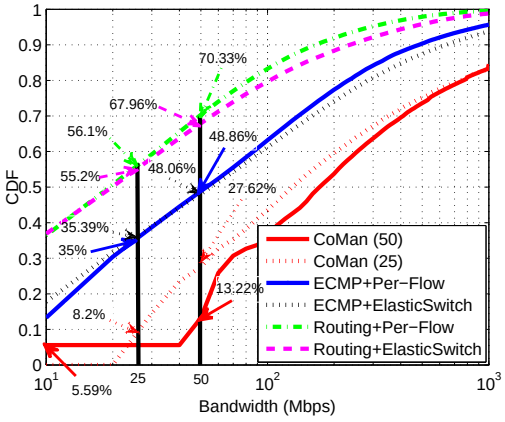
(b) Improvement on the average application completion time

Fig. 8. The performance related to application completion time when using CoMan, compared to ECMP+Per-Flow, ECMP+ElasticSwitch, Routing+Per-Flow, and Routing+ElasticSwitch.

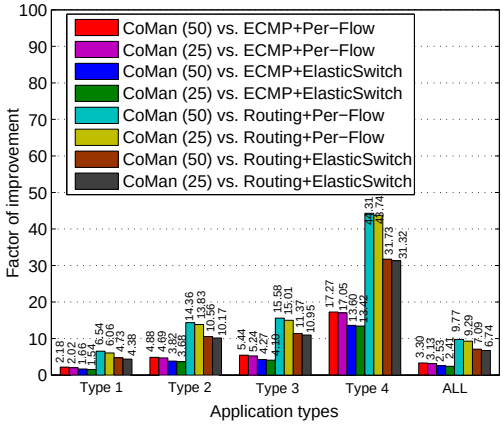
applications can be finished within 10s in the case of CoMan (25). Note that few applications failed (around 5.5%) in the case of CoMan (50). This is because that some computing frameworks cannot accommodate all applications on them, due to the high minimum bandwidth configured for each application.

Compared to ECMP+Per-Flow, ECMP+ElasticSwitch, Routing+Per-Flow, and Routing+ElasticSwitch, Fig. 8(b) shows that both CoMan (50) and CoMan (25) achieve smaller average ACTs. Note that, we only consider the applications, which are successfully admitted in the case of CoMan (50). The factor of improvement for the CoMan (50) is larger than that for the CoMan (25) across all application types. An interesting observation is that the computing framework with more application instances achieves less improvement on the average ACT. Such observation is mainly from the fact that the improvement on the average ACT of the four computing frameworks increases from Type 1 to Type 4 while the number of applications for them decreases from Type 1 to Type 4 (i.e., the number of applications for Type 1 to Type 4 are 76750, 31905, 28710, 13717 respectively). The underlying reason for this observation is that more applications can lead to more intense competition of the bandwidth. Compared to ECMP+Per-Flow, the improvement in average ACT for CoMan (50) and CoMan (25) is up to 14.41× and 11.67×, respectively. Whereas, compared

to ECMP+ElasticSwitch, we observe improvements of up to $19.38\times$ and $15.69\times$ for CoMan (50) and CoMan (25), respectively. Compared to Routing+Per-Flow, the improvements can be up to $47.61\times$ and $38.55\times$ for CoMan (50) and CoMan (25), respectively. Compared to Routing+ElasticSwitch, the improvements can be up to $68.75\times$ and $55.67\times$ for CoMan (50) and CoMan (25), respectively. Across all application types, the average ACT achieved by CoMan is improved by up to $4.55\times$, $6.68\times$, $16.32\times$, and $24.93\times$, compared to ECMP+Per-Flow, ECMP+ElasticSwitch, Routing+Per-Flow, and Routing+ElasticSwitch, respectively. In Fig. 8, we observe that the ElasticSwitch based schemes perform even worse than Per-Flow based schemes in the metric of ACT. The main reason for this is that even though ElasticSwitch is able to manage bandwidth well and allocate more bandwidth to the applications, compared to Per-Flow, however, ElasticSwitch may not be able to distribute the allocated bandwidth of an application to the individual network flows in an optimal way, as it is unaware of the application-level semantic. We can further observe that ECMP based schemes perform better than Routing based schemes. The is because that this routing selects the link with least workload among all links associated with a switch, making it unable to achieve global optimal strategy for balancing the traffic load among all links in the network. Eventually, this affects the ACTs of applications.



(a) CDF of per application bandwidth allocation



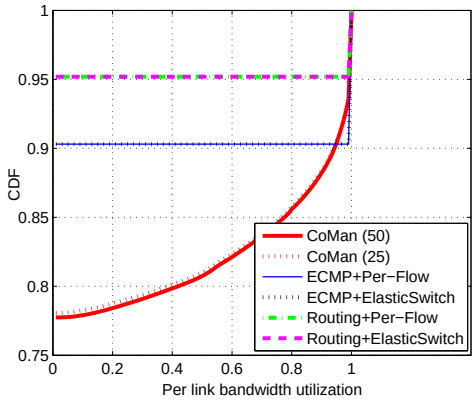
(b) Improvement on average bandwidth allocation

Fig. 9. Performance related to bandwidth allocation of application when using CoMan, compared to ECMP+Per-Flow, ECMP+ElasticSwitch, Routing+Per-Flow, and Routing+ElasticSwitch.

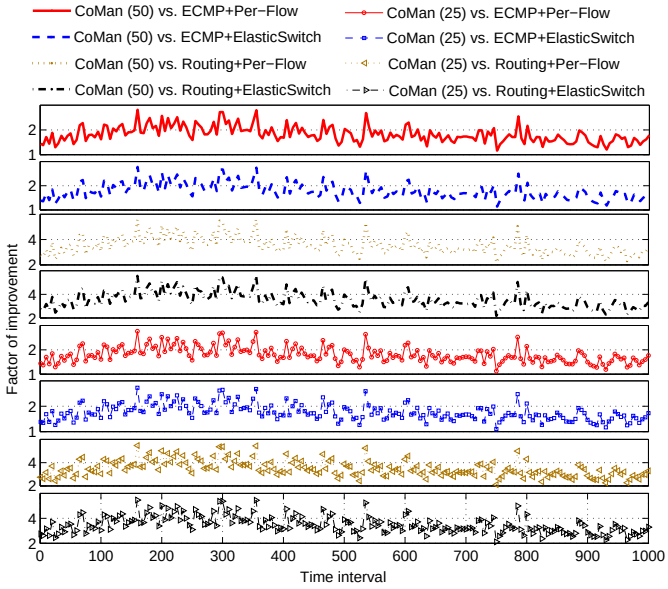
The bandwidth allocation of applications: To investigate the bandwidth allocated to each application, Fig. 9(a) plots the CDF of the bandwidth per application. Note that the X-axis is in logarithmic scale and the bandwidth allocated to the rejected applications is 0. We can see that about 5.59% of the applications obtain 0Mbps in CoMan (50). We further observe that most applications in CoMan (25) and CoMan (50) can acquire an amount of bandwidth greater than 25Mbps and 50Mbps, respectively. The portion of applications with bandwidth less than 25Mbps in CoMan (25) and 50Mbps in CoMan (50) is only 8.2% and 13.22%, respectively. On the contrary, 35% and 48.86% of applications obtain less than 25Mbps and 50Mbps in the case of ECMP+Per-Flow, respectively. ECMP+ElasticSwitch performs a little bit better than ECMP+Per-Flow, as the portion of applications with bandwidth less than 50Mbps is 48.06%. For Routing+Per-Flow, 56.1% and 70.33% of applications obtain less than 25Mbps and 50Mbps respectively. While those portions for Routing+ElasticSwitch are 55.2% and 67.96%.

Fig. 9(b) plots the factor of improvements on the average bandwidth allocation across all applications and all time intervals for both CoMan (50) and CoMan (25), compared to ECMP+Per-Flow and ECMP+ElasticSwitch. We can observe that as the number of applications of a computing framework increases, the factor of improvement decreases. This directly confirms the phenomenon in Fig. 8(b) that the more application instances in a computing framework, the less improvement on the average ACT. Compared to ECMP+Per-Flow and ECMP+ElasticSwitch, the improvement on the average application bandwidth achieved by CoMan (50) and CoMan (25) can be up to $17.27\times$ and $13.60\times$, respectively. Whereas, compared to Routing+Per-Flow and Routing+ElasticSwitch, we observe improvements of up to $44.31\times$ and $43.74\times$ for CoMan (50) and CoMan (25), respectively. Across all application types, CoMan also improves the average bandwidth by up to $3.30\times$, $2.53\times$, $9.77\times$ and $7.09\times$, compared to ECMP+Per-Flow, ECMP+ElasticSwitch, Routing+Per-Flow, and Routing+ElasticSwitch, respectively. We further observe that even though ECMP+ElasticSwitch can allocate more bandwidth to applications than ECMP+Per-Flow, it performs poor in terms of ACT, because of more skewed usage of the application bandwidth. Also, the reason for why Routing based schemes perform worse than ECMP based schemes is that the routing achieves worse load balancing than ECMP since it can only balance the load among the links related to a switch rather than the whole network.

Link bandwidth utilization: Fig. 10(a) shows the CDF of per link bandwidth utilization across all links over different time intervals. We observe that all schemes, ECMP+Per-Flow, ECMP+ElasticSwitch, Routing+Per-Flow and Routing+ElasticSwitch, cause the skewed use of link bandwidth. In other words, they fully utilize some links, while leave the other links to experience extremely low bandwidth utilization. More precisely, ECMP+Per-Flow and ECMP+ElasticSwitch fully utilize about 10% of links, and the reminder 90% of links are almost idle, while Routing+Per-Flow and Routing+ElasticSwitch can only fully utilize about 5% of links. On the other hand, CoMan utilizes link bandwidth more evenly and we observe that



(a) CDF of per link bandwidth utilization



(b) Factor of improvement on average link bandwidth utilization

Fig. 10. Performance related to link bandwidth utilization using CoMan, compared to ECMP+Per-Flow, ECMP+ElasticSwitch, Routing+Per-Flow, and Routing+ElasticSwitch.

CoMan (50) achieves a little bit higher link bandwidth utilization than CoMan (25). This is because the CoMan (50) configures a higher minimum guaranteed bandwidth for each application.

Fig. 10(b) shows the factor of improvement on average bandwidth utilization across all links in the network. We observe that both CoMan (50) and CoMan (25) perform better than ECMP+Per-Flow, ECMP+ElasticSwitch, Routing+Per-Flow and Routing+ElasticSwitch, in terms of the average link bandwidth utilization as the factor of improvement is always larger than 1. The main reason for low average bandwidth utilization of ECMP+Per-Flow, ECMP+ElasticSwitch, Routing+Per-Flow and Routing+ElasticSwitch, is that they make the skewed use of link bandwidth. We also observe that compared to ECMP+Per-Flow and ECMP+ElasticSwitch, the factor of improvement on the average link bandwidth utilization achieved by CoMan (50) is up to $2.88\times$ and $2.83\times$, respectively. While CoMan (25) improves performance by up to $2.83\times$ and $2.78\times$, respectively. Compared to Routing+Per-Flow and Routing+ElasticSwitch, the factors of improvement on the average link bandwidth utilization achieved by CoMan are both up to $5.59\times$.

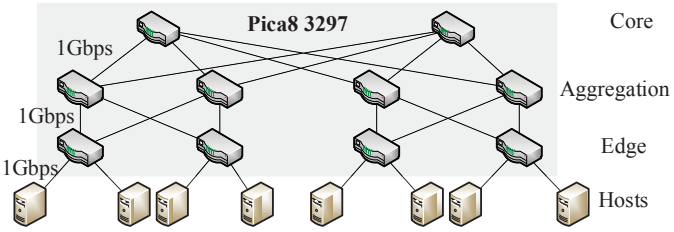


Fig. 11. Testbed topology.

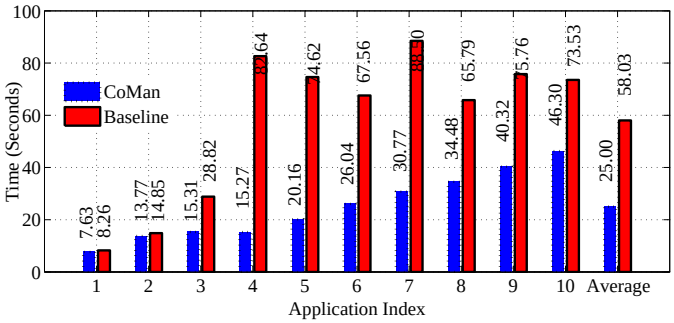
Remarks: The above results verify the three design principles of CoMan: *1)* elastic use of bandwidth among computing frameworks; *2)* guaranteeing the network performance of applications to speed up the completion; *3)* avoiding the skewed use of link bandwidth to improve link bandwidth utilization.

7.2 Small-scale testbed implementation

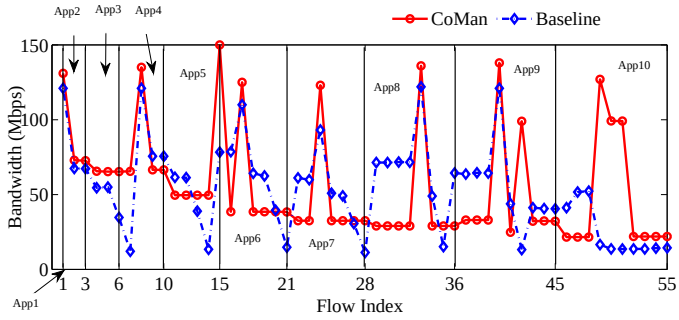
We embed our CoMan into a Software-defined Network (SDN) controller, which performs the bandwidth allocation and path selection strategies. For the path selection, we use the controller to configure the flow entries of involved switches along the flow's routing path. Meanwhile, we leverage SDN functions (e.g., the MeterTable in OpenFlow) to enforce the bandwidth allocated to each flow. In our implementation, we construct a datacenter testbed with a Fat-Tree like topology comprising 10 switches and 8 servers, as shown in Fig. 11. We use a Pica8 3297 48-port Gigabit switch with PicOS 2.6.32 system that supports both Layer 2/3 and OpenFlow. Each switch has 4 ports created from the virtualization of a SDN switch. Each server has a 2-core Intel(R) Pentium(R) 3.00GHz CPU, 2GB RAM, and 1G NICs. All the servers are configured with the Ubuntu, 12.04 64bit version operating system.

To evaluate the CoMan performance, we inject 10 applications with 55 flows into the datacenter network. These applications are considered to be submitted by two computing frameworks. The flow size is set to be 1Gb. We use iperf to generate TCP flows, and the traffic pattern of all the flows follows an all-to-all manner. We distribute these flows among the 10 applications, where each application a_j $(j{=}1, 2, \cdots, 10)$ contains j flows. For the parameters settings, the minimum guaranteed bandwidth and the maximum reservable bandwidth for each computing framework accounts for 50% and 60%, respectively, on each VLG. In addition, for all the computing frameworks and all VLGs, we use $\alpha_{i,k}{=}100$Mbps and $\beta_{i,k}{=}200$Mbps to enforce the bandwidth per application. As a comparison, we also evaluate the case of the baseline, where all the flows are routed by the default shortest path, and all of them fairly compete for bandwidth. That is, the baseline does not use CoMan.

Fig. 12(a) shows that the CoMan can significantly reduce (e.g., up to 67.37 seconds for the 4-th applications) the ACT of all 10 applications, in the testbed experiment. Across all applications, the average reduction of the ACT is 33.03 seconds resulting in the factor of improvement on the ACT achieved by CoMan of up to $2.32\times$. To understand this on a microscopic level, we plot the bandwidth allocated to the individual flows in Fig. 12(b). We can see that the



(a) Application completion time in CoMan and the baseline.



(b) Bandwidth allocation to the individual flows in CoMan and the baseline.

Fig. 12. Performance comparison in the test bed deployment.

minimum value of the flows' bandwidth of one application in the absence of CoMan is always lower than that in the case with CoMan. The experimental results demonstrate that CoMan can account for the collective behaviors of flows when performing the bandwidth allocation, and can adequately utilize the link bandwidth, given the same traffic load, which improves the application completion times.

8 Discussion

Supporting other datacenter network topologies: So far we have only focused on the Fat-Tree topology. However, it is important to keep in mind that our solution can simply be extended to other topologies [50, 51]. For example, when extending our solution to BCube [51], the only thing we need to do is to construct VLG groups. For example, in BCube, the links that source from, or direct to a same server can be encapsulated into a VLG. With the VLGs in BCube, we can then construct the VLG dependency graph and perform the bandwidth allocation and routing with the same way we did in this paper. We leave this as one part of our future work.

Incorporating with existing computing frameworks: CoMan relies on bandwidth allocation and routing techniques to be deployed in reality, and both techniques can be implemented by taking advantages of the SDN controllers and SDN switches. To incorporate CoMan with existing computing frameworks, the only thing we need to do is to design a middle layer that can bridge the computing frameworks and the SDN controller. For example, the SDN controller can get the flow information of Spark applications through the `MapOutputTracker` and the Spark DAG scheduler [52, 53]. We remain this point as an open challenge.

Source/destination placement: CoMan assumes that the source and destination nodes are fixed when performing bandwidth allocation and routing for each flow. This simplifies the complexity of managing the bandwidth across multiple computing frameworks in a datacenter. However, in general, the relations among source/destination placement (e.g., data placement, map/reduce placement), bandwidth allocation and routing are deeply intertwined with each other. This can be incorporated into CoMan by using the data/task placement techniques in [54, 55] before we perform bandwidth allocation and routing for the network flows. In such a case, the application performance can further be improved. We leave this as another part of our future work.

Dealing with WAN bandwidth allocation: Currently, CoMan focuses only on managing the in-network bandwidth within a single datacenter. Since the data-parallel jobs are increasingly running across multiple geographically distributed datacenters [56, 57], one may question that can CoMan be extended to manage the inter-datacenter WAN bandwidth? Actually, this can incorporated into CoMan by first encapsulating the inter-datacenter links source from or direct to a same datacenter into a VLG group and then constructing VLG dependency graph. Finally, we can perform bandwidth allocation with the resources in each VLG and also select routing paths for each flow based on the VLG dependency graph.

Handling applications within one computing framework: One may wonder at this point that even when deploying one computing framework in a datacenter, the bandwidth utilization "skewness" still exists. This is because that such "skewness" is caused by the skew input data of applications [35–37], rather than deploying different computing frameworks in a datacenter. In such a case, how to manage the bandwidth? One possible way is to define a set of priority queues, and leverage the *max-min bandwidth constraint model* to set the bandwidth that each queue can use. We can then distribute the bandwidth of each queue to its applications and flows, and finally perform routing for each flow.

9 Conclusion

In this paper, we propose a bandwidth management architecture called CoMan, which enables the global coordination among multiple computing frameworks for achieving high bandwidth utilization and short application completion times. Specifically, we tackle a challenging link coupling problem, and propose a novel abstraction of VLGs to virtualize the bandwidth into a pool. Accordingly, we introduce a three-level bandwidth allocation model to allow elastic bandwidth sharing among computing frameworks as well as guarantee the network performance for applications. We further propose a novel VLG dependency graph and formulate a minimum residual bandwidth problem to guide the path selection, with high bandwidth utilization as the objective. Overall, the trace-driven simulation results show that CoMan improves the bandwidth utilization and speeds up the application completion time by up to $2.83\times$ and $6.68\times$, respectively, in comparison to the widely used ECMP+ElasticSwitch solution. Moreover, the testbed results show that applications complete up to $2.32\times$ faster, on average, when using CoMan.

ACKNOWLEDGMENT

This work is supported by the National Key Research and Development Program of China No. 2016YFB1000205; the State Key Program of National Natural Science of China under Grant 61432002; the NSFC under Grant 61672379, Grant 61772112 and Grant 61370199; the Dalian High-level Talent Innovation Program under Grant 2015R049; the National Natural Science Foundation for Outstanding Excellent young scholars of China under Grant 61422214; National Natural Science Foundation of China under Grant 61772544; National Basic Research Program (973 program) under Grant 2014CB347800.

REFERENCES

[1] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.
[2] "Spark," http://spark.apache.org/.
[3] G. Malewicz, M. H. Austern, A. J. C. Bik, J. C. Dehnert, I. Horn, L. Naty, and G. Czajkowski, "Pregel: A system for large-scale graph processing," in *Proc. of ACM SIGMOD*, 2010.
[4] A. Greenberg, J. Hamilton, D. A. Maltz, and P. Patel, "The cost of a cloud: research problems in data center networks," *ACM SIGCOMM Computer Communication Review*, vol. 39, no. 1, pp. 68–73, 2008.
[5] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. D. Joseph, R. H. Katz, S. Shenker, and I. Stoica, "Mesos: A platform for fine-grained resource sharing in the data center." in *Proc. of USENIX NSDI*, 2011.
[6] M. Chowdhury, M. Zaharia, J. Ma, M. I. Jordan, and I. Stoica, "Managing data transfers in computer clusters with orchestra," in *Proc. of ACM SIGCOMM*, 2011.
[7] W. Li, X. Yuan, K. Li, H. Qi, and X. Zhou, "Leveraging endpoint flexibility when scheduling coflows across geo-distributed datacenters," in *Proc. of IEEE INFOCOM*, 2018.
[8] M. Schwarzkopf, A. Konwinski, M. Abd-El-Malek, and J. Wilkes, "Omega: flexible, scalable schedulers for large compute clusters," in *Proc. of ACM EuroSys*, 2013.
[9] V. K. Vavilapalli, A. C. Murthy, C. Douglas, S. Agarwal, M. Konar, R. Evans, T. Graves, J. Lowe, H. Shah, S. Seth, B. Saha, C. Curino, O. O'Malley, S. Radia, B. Reed, and E. Baldeschwieler, "Apache hadoop YARN: yet another resource negotiator," in *Proc. of ACM SOCC*, 2013.
[10] M. Chowdhury, Y. Zhong, and I. Stoica, "Efficient coflow scheduling with varys," in *Proc. of ACM SIGCOMM*, 2014.
[11] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat, "Hedera: Dynamic flow scheduling for data center networks," in *Proc. of USENIX NSDI*, 2010.
[12] 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 ACM SIGCOMM*, 2013.
[13] A. Shieh, S. Kandula, A. Greenberg, C. Kim, and B. Saha, "Sharing the data center network," in *Proc. of USENIX NSDI*, Boston, America, 2011.
[14] K. He, E. Rozner, K. Agarwal, Y. J. Gu, W. Felter, J. Carter, and A. Akella, "Ac/dc tcp: Virtual congestion control enforcement for datacenter networks," in *Proc. of ACM SIGCOMM*, 2016.
[15] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," in *Proc. of ACM SIGCOMM*, 2008.
[16] Y. Zhao, K. Chen, W. Bai, C. Tian, Y. Geng, Y. Zhang, D. Li, and S. Wang, "Rapier: Integrating routing and scheduling for coflow-aware data center networks," in *Proc. of IEEE INFOCOM*, 2015.
[17] B. Briscoe, "Flow rate fairness: Dismantling a religion," *ACM SIGCOMM Computer Communication Review*, vol. 37, no. 2, pp. 63–74, 2007.
[18] L. Popa, G. Kumar, M. Chowdhury, A. Krishnamurthy, S. Ratnasamy, and I. Stoica, "Faircloud: sharing the network in cloud computing," in *Proc. of ACM SIGCOMM*, 2012.
[19] J. Guo, F. Liu, D. Zeng, J. Lui, and H. Jin, "A cooperative game based allocation for sharing data center networks," in *Proc. of IEEE INFOCOM*, 2013.
[20] J. Guo, F. Liu, J. C.S.Lui, and J. Hai, "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.
[21] 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.
[22] J. Guo, F. Liu, T. Wang, and J. C. Lui, "Pricing intra-datacenter networks with over-committed bandwidth guarantee," in *Proc. of Usenix ATC*, 2017.
[23] T. Lam, S. Radhakrishnan, A. Vahdat, and G. Varghese, "Netshare: Virtualizing data center networks across services," University of California, San Diego, Tech. Rep., 2010.
[24] L. Chen, Y. Feng, B. Li, and B. Li, "Towards performance-centric fairness in datacenter networks," in *Proc. of IEEE INFOCOM*, 2014.
[25] T. Benson, A. Anand, A. Akella, and M. Zhang, "Microte: Fine grained traffic engineering for data centers," in *Proc. of ACM CoNext*, 2011.
[26] M. Alizadeh, S. Yang, M. Sharif, S. Katti, N. McKeown, B. Prabhakar, and S. Shenker, "pfabric: Minimal near-optimal datacenter transport," in *Proc. of ACM SIGCOMM*, 2013.
[27] A. Munir, G. Baig, S. M. Irteza, I. A. Qazi, A. X. Liu, and F. R. Dogar, "Friends, not foes: synthesizing existing transport strategies for data center networks," in *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 4, 2014, pp. 491–502.
[28] A. Munir, I. A. Qazi, Z. A. Uzmi, A. Mushtaq, S. N. Ismail, M. S. Iqbal, and B. Khan, "Minimizing flow completion times in data centers," in *Proc. of IEEE INFOCOM*, 2013.
[29] W. Bai, L. Chen, K. Chen, D. Han, C. Tian, and H. Wang, "Information-agnostic flow scheduling for commodity data centers," in *Proc. of NSDI*, 2015.
[30] F. R. Dogar, T. Karagiannis, H. Ballani, and A. Rowstron, "Decentralized task-aware scheduling for data center networks," in *Proc. of ACM SIGCOMM*, 2014.
[31] M. Chowdhury and I. Stoica, "Efficient coflow scheduling without prior knowledge," in *Proc. of ACM SIGCOMM*, 2015.
[32] M. Isard, V. Prabhakaran, J. Currey, U. Wieder, K. Talwar, and A. Goldberg, "Quincy: fair scheduling for distributed computing clusters," in *Proc. of ACM SOSP*, 2009.
[33] M. Zaharia, D. Borthakur, J. Sen Sarma, K. Elmeleegy, S. Shenker, and I. Stoica, "Delay scheduling: a simple technique for achieving locality and fairness in cluster scheduling," in *Proc. of ACM EuroSys*, 2010.
[34] A. Munir, T. He, R. Raghavendra, F. Le, and A. X. Liu, "Network scheduling aware task placement in datacenters," in *Proc. of ACM International on Conference on emerging Networking EXperiments and Technologies*, 2016.
[35] Y. Le, J. Liu, F. Ergun, and D. Wang, "Online load balancing for mapreduce with skewed data input," in *Proc. of IEEE INFOCOM*, 2014.
[36] Y. Kwon, M. Balazinska, B. Howe, and J. Rolia, "Skewtune: Mitigating skew in mapreduce applications," in *Proc. of ACM SIGMOD*, 2012.
[37] S. R. Ramakrishnan, G. Swart, and A. Urmanov, "Balancing reducer skew in mapreduce workloads using progressive sampling," in *Proc. of ACM SoCC*, 2012.
[38] B. Heller, S. Seetharaman, P. Mahadevan, Y. Yiakoumis, P. Sharma, S. Banerjee, and N. McKeown, "Elastictree: Saving energy in data center networks," in *Proc. of Usenix NSDI*, 2010.
[39] X. Wang, Y. Yao, X. Wang, K. Lu, and C. Qing, "Carpo: Correlation-aware power optimization in data center networks," in *Proc. of IEEE INFOCOM*, 2012.
[40] K. Zheng, X. Wang, L. Li, and X. Wang, "Joint power optimization of data center network and servers with correlation analysis," in *Proc. of IEEE INFOCOM*, 2014.
[41] H. Zhang, J. Zhang, W. Bai, K. Chen, and M. Chowdhury, "Resilient datacenter load balancing in the wild," in *Proc. of ACM SIGCOMM*, 2017.
[42] P. Bodík, I. Menache, M. Chowdhury, P. Mani, D. A. Maltz, and I. Stoica, "Surviving failures in bandwidth-constrained datacenters," in *Proc. of ACM SIGCOMM*, 2012.
[43] T. Wood, K. Ramakrishnan, P. Shenoy, J. Van der Merwe, J. Hwang, G. Liu, and L. Chaufournier, "Cloudnet: Dynamic pooling of cloud resources by live wan migration of virtual machines," *IEEE/ACM Transactions on Networking*, vol. 23, no. 5, pp. 1568–1583, 2015.
[44] "Amazon elastic computing cloud," http://aws.amazon.
[45] L. Luo, D. Guo, J. Wu, T. Qu, T. Chen, and X. Luo, "Vlccube:

A vlc enabled hybrid network structure for data centers," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 7, pp. 2088–2102, 2017.
[46] S. Hu, K. Chen, H. Wu, W. Bai, C. Lan, H. Wang, H. Zhao, and C. Guo, "Explicit path control in commodity data centers: Designs and applications," in *Proc. of USENIX NSDI*, 2015.
[47] Y. Feng, B. Li, and B. Li, "Bargaining towards maximized resource utilization in video streaming datacenters," in *Proc. of IEEE INFOCOM*, 2012.
[48] D. Li, Y. Shang, and C. Chen, "Software defined green data center network with exclusive routing," in *Proc. of IEEE INFOCOM*, 2014.
[49] C. Reiss, J. Wilkes, and J. Hellerstein, "Google cluster-usage traces," http://code.google.com/p/googleclusterdata.
[50] D. Guo, C. Li, J. Wu, and X. Zhou, "Dcube: A family of network structures for containerized data centers using dual-port servers," *Elsevier Computer Communications*, vol. 53, pp. 13–25, 2014.
[51] C. Guo, G. Lu, D. Li, H. Wu, X. Zhang, Y. Shi, C. Tian, Y. Zhang, and S. Lu, "Bcube: a high performance, server-centric network architecture for modular data centers," in *Proc. of the ACM SIGCOMM*, 2009.
[52] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, and I. Stoica, "Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing," in *Proc. of USENIX NSDI*, 2012.
[53] Z. Hu, B. Li, and J. Luo, "Flutter: Scheduling tasks closer to data across geo-distributed datacenters," in *Proc. of IEEE INFOCOM*, 2016.
[54] M. Chowdhury, S. Kandula, and I. Stoica, "Leveraging endpoint flexibility in data-intensive clusters," in *Proc. of ACM SIGCOMM*, 2013.
[55] V. Jalaparti, P. Bodik, I. Menache, S. Rao, K. Makarychev, and M. Caesar, "Network-aware scheduling for data-parallel jobs: Plan when you can," in *Proc. of ACM SIGCOMM*, 2015.
[56] Q. Pu, G. Ananthanarayanan, P. Bodik, S. Kandula, A. Akella, P. Bahl, and I. Stoica, "Low latency geo-distributed data analytics," in *Proc. of ACM SIGCOMM*, 2015.
[57] R. Viswanathan, G. Ananthanarayanan, and A. Akella, "Clarinet: Wan-aware optimization for analytics queries," in *Proc. of USENIX OSDI*, 2016.

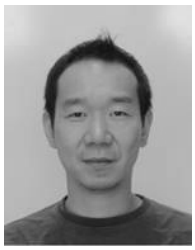
Wenxin Li received the B.E. degree from the School of Computer Science and Technology, Dalian University of Technology, China, in 2012. Currently, he is a Ph.D. candidate in the School of Computer Science and Technology, Dalian University of Technology, China. His research interests include datacenter networks and cloud computing.

Deke Guo received the B.S. degree in industry engineering from Beijing University of Aeronautic and Astronautic, Beijing, China, in 2001, and the Ph.D. degree in management science and engineering from National University of Defense Technology, Changsha, China, in 2008. He is an Associate Professor with the College of Information System and Management, National University of Defense Technology, Changsha, China. His research interests include distributed systems, software-defined networking, datacenter networking, wireless and mobile systems, and interconnection networks.

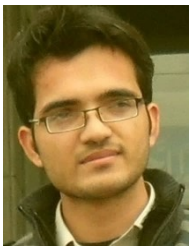
Alex X. Liu received the Ph.D. degree in computer science from The University of Texas at Austin in 2006. His research interests focus on networking and security. He received the IEEE & IFIP William C. Carter Award in 2004, the National Science Foundation CAREER Award in 2009, and the Michigan State University Withrow Distinguished Scholar Award in 2011. He received the best paper awards from ICNP-2012, SRDS-2012, and LISA-2010. He is currently an Associate Editor of the IEEE/ACM TRANSACTIONS ON NETWORKING, an Editor of the IEEE TRANSACTIONS ON DEPENDABLE AND SECURE COMPUTING, and an Area Editor of Computer Communications.

Keqiu Li received the bachelors and masters degrees from the Department of Applied Mathematics at the Dalian University of Technology in 1994 and 1997, respectively. He received the Ph.D. degree from the Graduate School of Information Science, Japan Advanced Institute of Science and Technology in 2005. He also has two-year postdoctoral experience in the University of Tokyo, Japan. He is currently a professor in the School of Computer Science and Technology, Dalian University of Technology, China. He has published more than 100 technical papers, such as IEEE TPDS, ACM TOIT, and ACM TOMCCAP. He is an Associate Editor of IEEE TPDS and IEEE TC. He is a senior member of IEEE. His research interests include internet technology, datacenter networks, cloud computing and wireless networks.

Heng Qi was a Lecture at the School of Computer Science and Technology, Dalian University of Technology, China. He got bachelor's degree from Hunan University in 2004 and master's degree from Dalian University of Technology in 2006. He servered as a software engineer in GlobalLogic-3CIS from 2006 to 2008. Then he got his doctorate degree from Dalian University of Technology in 2012. His research interests include computer network, multimedia computing, and mobile cloud computing. He has published more than 20 technical papers in international journals and conferences, including ACM Transactions on Multimedia Computing, Communications and Applications (ACM TOMCCAP) and Pattern Recognition (PR).



Song Guo received his Ph.D. in computer science from University of Ottawa. He is currently a full professor at Department of Computing, The Hong Kong Polytechnic University. Prior to joining PolyU, he was a full professor with the University of Aizu, Japan. His research interests are mainly in the areas of cloud and green computing, big data, wireless networks, and cyber-physical systems. He has published over 300 conference and journal papers in these areas and received multiple best paper awards from IEEE/ACM conferences. His research has been sponsored by JSPS, JST, MIC, NSF, NSFC, and industrial companies. Dr. Guo has served as an editor of several journals, including IEEE Transactions on Parallel and Distributed Systems (2011-2015), IEEE Transactions on Emerging Topics in Computing (2013-), IEEE Transactions on Green Communications and Networking (2016-), IEEE Communications Magazine (2015-), and Wireless Networks (2013-). He has been actively participating in international conferences as general chair and TPC chair. He is a senior member of IEEE, a senior member of ACM, and an IEEE Communications Society Distinguished Lecturer.



Ali Munir is a PhD. student at Michigan State University at Computer Science & Engineering Dept. He received his BS in Electronics Engineering and MS in Electrical Engineering from National University of Sciences and Technology NUST, Pakistan. His research interests focus on networking and security.

Xiaoyi Tao received the bachelor's degree from the school of Software Engineering, Dalian University of Technology, China, in 2011. Currently, she is a Ph.D. candidate in the School of Computer Science and Technology, Dalian University of Technology, China. From December 2016, she is a Visiting Scholar in the Emerging Networks and Systems Lab and Wireless Networks Lab, the Department of Information and Electronic Engineering, Muroran Institute of Technology, Muroran, Hokkaido, Japan, under the guidance of Prof. Kaoru Ota. Her research interests include datacenter networks, SDN networks and cloud computing.