TeaVisor: Network Hypervisor for Bandwidth Isolation in SDN-NV

Yeonho Yoo, *Graduate Student Member, IEEE*, Gyeongsik Yang, *Member, IEEE*, Jeunghwan Lee, Changyong Shin, Hoseok Kim, and Chuck Yoo, *Member, IEEE*

Abstract—We introduce TeaVisor that provides bandwidth isolation guarantee for network virtualization (NV) based on software-defined networking (SDN). SDN-based NV (SDN-NV) offers many benefits to clouds, such as topology and address virtualization while allowing flexible resource provisioning, control, and monitoring on virtual networks. In SDN-NV, however, routing is done by tenants independently; thus, existing studies have difficulties in bandwidth isolation guarantee due to the overloaded link problem. Bandwidth isolation guarantee is essential for providing stable and reliable throughput on network services in SDN-NV. Without bandwidth isolation guarantee, tenants suffer degraded service qualities and significant loss in revenue. To address this problem, we design and implement TeaVisor in three components: path virtualization, bandwidth reservation, and path establishment. Through extensive experiments, TeaVisor shows that bandwidth isolation is guaranteed with near-zero errors, which is three orders of magnitude better than existing studies. In addition, TeaVisor guarantees the minimum and maximum bandwidth at the same time. We also present an overhead analysis of TeaVisor in control traffic and memory consumption.

Index Terms—Network communication, network management, network operating systems

1 Introduction

Software-defined networking (SDN) has made network systems open and softwarized and has been evolving in many directions, such as network function virtualization (NFV) [2] and transport SDN (T-SDN) [3]. Among them, SDN-based network virtualization (SDN-NV) combines network virtualization (NV) with SDN [4], [5], [6], [7], [8], [9]. The need for SDN-NV arises from datacenters, in which tenant networks must be isolated via virtual networks (VNs). Each tenant has its own SDN controller[1] that receives and sends control messages, such as new packet generation or flow rule installation. An advantage of SDN-NV is that the tenant can create its own VN topology on top of the physical network, and the SDN controller can calculate its own packet-forwarding paths between entities [10], [11].[2] With such benefits, a tenant can constitute a custom VN topology of network switches and monitor for their performance bottleneck analysis [12]. Additionally, within its VN, the tenant can process network connections with priorities for the purpose of quality-of-experience enhancements [9].

Datacenter applications exist on a broad scale, which ranges from commodity applications (e.g., video streaming and distributed file systems) to machine learning workloads (e.g., training and inference of neural networks [13], [14]). Many of these applications require network performance isolation (i.e., multiple tenants on a shared network should be able to use the expected network bandwidth[3]) [15]. Performance isolation of multiple tenants is typically expressed in bandwidth requirements for the services of tenants [16], [17]. When these requirements are not satisfied, the quality of the respective services is severely damaged [18], and a significant loss in revenue occurs [19], [20]. Therefore, the bandwidth isolation guarantee is a critical problem, yet SDN-NV poses significant challenges to supporting bandwidth isolation guarantee.

SDN-NV allows SDN controllers of tenants to calculate paths for the entity pairs, making it difficult for the existing studies on bandwidth isolation guarantee to be applied. Most existing studies [19], [20], [21], [22] do not consider routing, which determines the path between an entity pair, but instead use the hose model, in which entities are

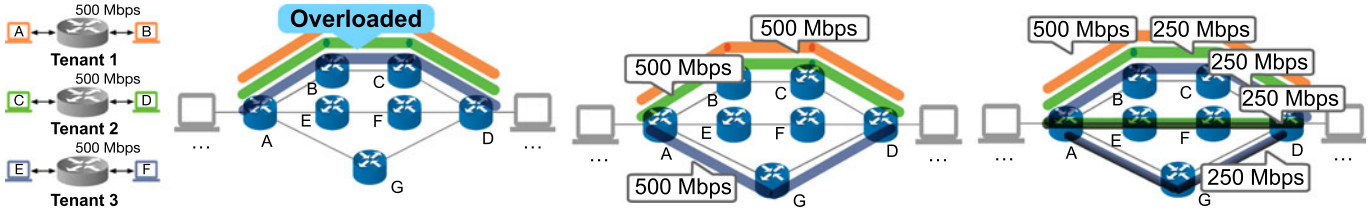
1. The term "SDN controller" in this paper refers to one operated by a tenant.

• *The authors are with the Department of Computer Science and Engineering, Korea University, Seoul 02841, South Korea. E-mail: {yhyoo, ksyang, jhlee21, cyshin, chuckyoo}@os.korea.ac.kr, thomas1234@korea.ac.kr.*

Manuscript received 16 June 2022; revised 1 November 2022; accepted 26 November 2022. Date of publication 1 December 2022; date of current version 6 September 2023.
This work was supported by Institute of Information & communications Technology Planning & Evaluation grant funded by the Korea government (Ministry of Science and ICT, MSIT) (2015-0-00280, (SW Starlab) Next generation cloud infrasoftware toward the guarantee of performance and security SLA). This research was also partly supported by Basic Science Research Program through the National Research Foundation of Korea funded by the Ministry of Education (NRF-2021R1A6A1A13044830), and a Korea University Grant. (Corresponding authors: Gyeongsik Yang and Chuck Yoo.)
Recommended for acceptance by K. Chen.
Digital Object Identifier no. 10.1109/TCC.2022.3225915

2. Without any explicit designation, we use the term "entity" to indicate the user's computing entities, such as virtual machines (VMs) and containers. So, entity pair means the pair consisting of source and destination entities.

3. We use the term "bandwidth" to indicate the maximum amount of data that can flow through a path or link. "Throughput" is used to indicate the actual amount of bandwidth consumed. Except in particular circumstances, the two terms are interchangeable.



(a) Overloaded link problem in SDN-NV. (b) Existing studies with the hose model. (c) Solution provided by TeaVisor.

Fig. 1. Bandwidth isolation guarantee examples.

connected via a non-blocking virtual switch. This approach assumes that the path is given (solved by an orthogonal method) and focuses on entity pair bandwidth. In SDN-NV, when multiple controllers calculate their own paths, each path has its own respective bandwidth requirement. This means that some links associated with the paths can be overloaded beyond the physical link capacity. We refer to this problem as the "overloaded link problem."

For example, in Fig. 1a, suppose that three tenants (tenants 1, 2, and 3) have one entity pair each and have their bandwidth requirements—500 Mbps of traffic for each entity pair (Fig. 1a, left), where all physical links can transmit 1 Gbps traffic. In SDN-NV, as the SDN controllers do not know each other, the controllers of all three tenants 1, 2, and 3 can possibly calculate the same forwarding paths, which consist of switches A-B-C-D in Fig. 1a. In this situation, the physical links in the path become overloaded because the total bandwidth requirements (1.5 Gbps) exceed the link capacity (1 Gbps). Therefore, the requested bandwidth requirements cannot be satisfied, even if the other physical paths (i.e., A-E-F-D and A-G-D) are available.

For the hose model in Fig. 1b, suppose that three tenants have the same bandwidth requirements and physical link capacity as in Fig. 1a. Then, assume that the paths of tenants 1, 2, and 3 resulting from the orthogonal routing methods are A-B-C-D, A-B-C-D, and A-G-D, respectively. Then, the existing studies make each entity pair consume the bandwidth per their bandwidth requirements, so for tenants 1 and 2, which share one path, each uses the path at 500 Mbps, and tenant 3 uses another path with 500 Mbps. In this case, all tenants' bandwidth requirements are satisfied. However, in SDN-NV, the hose model cannot address the overloaded link problem, because tenant controllers determine their routing path without considering other tenants.

Some of the existing studies are not based on the hose model but provide fair sharing between the source and the destination entities [16], [19], [23], [24]. For example, Seawall [16] pairs each source-end entity with a weight so that the share of bandwidth for the end entity is proportional to the paired weight in all network links. Here, the weight is given by the administrator, so the purpose of weight is to enable the fair sharing of network traffic and to control misbehaving traffic. For example, the bandwidth provided to an end entity can be arbitrarily reduced by the weights of other network traffic [20], so the bandwidth isolation guarantee cannot be achieved either.

Furthermore, existing studies on non-virtualized SDN, such as Predictor [25] and DBS [26], are also not directly related to the bandwidth isolation in SDN-NV. Predictor is used to minimize the flow table size to provide the predictable performance of switches and thereby of flows. DBS focuses on scheduling within switches that dynamically allocate data rates to flows. These methods implement rate limiting within switches. Therefore, their algorithms do not address the bandwidth isolation in SDN-NV.

Thus, we propose TeaVisor to provide the bandwidth isolation guarantee for SDN-NV. The key idea of TeaVisor is to associate the flow rules from the SDN controller with bandwidth requirements and to enable multipath routing to satisfy the bandwidth requirements. For example, as a solution to the problem in Fig. 1a, TeaVisor establishes additional paths for the bandwidth isolation guarantee (Fig. 1c) as follows. For the same path, A-B-C-D, that all tenants share, tenant 1 is allocated its required 500 Mbps, thus occupying half of the link capacity, while tenants 2 and 3 are allocated half of the bandwidth requirement each, 250 Mbps. For the remaining bandwidth requirements, 250 Mbps for tenants 2 and 3, TeaVisor establishes additional paths (via path virtualization to be explained later) and satisfies their requirements (e.g., path A-E-F-D for the 250 Mbps of tenant 2 and A-G-D for the 250 Mbps of tenant 3). In fact, various studies for multipath routing exist [27], [28], [29], [30], but they are not relevant to SDN-NV. We discuss their limitations in Section 2.3. To the best of our knowledge, TeaVisor is the first study for SDN-NV that considers multipath routing for the bandwidth isolation guarantee.

To make it practically meaningful, TeaVisor is designed to accept two types of bandwidth requirements: minimum and maximum, as these are the typical requirements in datacenters [15]. The minimum bandwidth requirement is the amount of bandwidth to be reserved for a path at all times. The maximum bandwidth requirement is the upper limit by which TeaVisor can exceed the minimum bandwidth requirement using the idle bandwidth. In a typical datacenter, tenants follow the pay-per-use policy, so it is important to adhere to the upper limit of the allocated bandwidth. However, according to [31], most datacenters could not guarantee the upper bandwidth limit, so tenants are in charge of much higher bandwidth. Thus, our goal of TeaVisor is to make sure that the minimum and maximum bandwidth requirements are both satisfied.

The bandwidth isolation guarantee of TeaVisor is accomplished using three components: path virtualization (Section 3.2), bandwidth reservation (Section 3.3), and path establishment (Section 3.4). First, path virtualization is for crafting the intended virtual path from the flow rules and for creating the physical paths that realize the virtual path in the physical network. Then, the bandwidth reservation

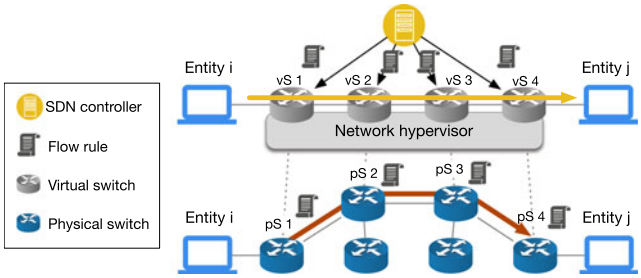


Fig. 2. Relationship between a path and flow rules.

component reserves the bandwidth requirement on physical paths at a link level.[4] Third, path establishment substantiates the physical paths through the flow rules, such as for packet splitting on multiple paths and for rate-limiting.

TeaVisor is implemented as a complete system based on Libera [9], an open-source network hypervisor (NH). We conduct comprehensive evaluations, including the satisfaction of bandwidth isolation and overheads with varying amounts of bandwidth requirements and numbers of tenants. In brief, TeaVisor achieves the following contributions:

- Provide the first bandwidth isolation guarantee for SDN-NV.
- Design path virtualization and bandwidth reservation for multipath routing in SDN-NV.
- Achieve near-zero error rates on the bandwidth isolation guarantee.

2 Background and Related Work

2.1 SDN-NV

SDN-NV virtualizes tenant networks via an NH. The NH [4], [5], [6], [9], [33], [34] abstracts the underlining (physical) network as VNs and provides them to SDN controllers. Compared to SDNs, SDN-NVs have two main characteristics: topology and address virtualization. Topology virtualization allows tenants to configure their arbitrary VN topology, which can differ significantly from the physical network. By using the NH, the number of switches, the number of ports, and the connections of the links that comprise a VN topology can be freely configured. When the VN is established and the switches in the VN boot, the NH connects with the SDN controllers of the tenants, as if the connection is made from an ordinary SDN switch. Therefore, SDN controllers can operate as though they control a physical network [9]. In addition, through address virtualization, the NH provides a virtual address that allows each tenant to freely choose their addressing scheme without concerning about conflicts with other tenants.

Fig. 2 explains the structure of SDN-NV: the top portion represents the VN and the bottom represents the physical network. Each VN consists of end entities (e.g., entities i and j), virtual switches (vS 1, 2, 3, and 4), and virtual links that connect virtual switches. When new network traffic arrives, an event is delivered to the SDN controller through the NH. Then, the SDN controller calculates a path (routing) between entities for the traffic (e.g., vS1 to vS4 in Fig. 2) and then sends flow rules (for individual virtual switches) to the NH. Upon receiving the flow rules, the NH translates and installs each flow rule while considering the address of the entities and the virtual topology of the mapped physical switches (pS1 to pS4). This example shows that all network controls and operations are based on flow rules translation by the NH.

So far, various NHs have been proposed. Table 1 summarizes the differences between these NHs, including the focus aspect, where virtualization is applied, and whether bandwidth isolation is guaranteed. First, we explain the focus aspect. Existing NHs improved various aspects of SDN-NV as follows. FlowVisor [4] was the first NH to share SDN switches with multiple tenants. FlowN [5] proposed a more scalable architecture of NH using containers. OpenVirteX [6] is an open-source NH that extended FlowVisor with improved address virtualization. CoVisor [7] merged flow rules to enable multiple SDN controllers to manage them as a single network. Libera [9] proposed a datacenter service model of SDN-NV. V-Sight [32] supported network monitoring in SDN-NV by providing isolated statistics for VNs.

Second, we compare where virtualization is applied. For topology virtualization, FlowVisor partially supported the topology virtualization as it creates a virtual switch by one-to-one mapping with the physical switch. On the other hand, FlowN supported the arbitrary mapping of virtual switches to physical switches, which was not possible in FlowVisor. So FlowN fully supported topology virtualization. Other NHs supported topology virtualization similar to that of FlowN. In terms of address virtualization, FlowVisor did not virtualize addresses but split the portion of addresses to tenants. FlowN introduced VLAN-based address virtualization to provide an entire address space for each tenant. OpenVirteX provided address mapping between virtual networks and the physical network. The other NHs followed the address virtualization schemes of either FlowN or OpenVirteX. None of these existing NHs provided path virtualization. The lack of path virtualization hinders multipath routing and the bandwidth isolation guarantee.

Third, FlowVisor attempted to isolate the bandwidth of each tenant via multiple priority queues at each switch, which is a common feature of existing switches. However, by mapping the traffic of end pairs to each queue, FlowVisor only achieved bandwidth management per queue. A typical tenant creates tens or hundreds of entities [36]; thus, the number of entity pairs easily exceeds the number of queues. Therefore, FlowVisor provided only limited bandwidth isolation because the switch has a limited number of queues (e.g., two to eight) [37]. In summary, to the best of our knowledge, no NH has yet solved the bandwidth isolation guarantee problem. Therefore, TeaVisor is the first NH to do so for each entity pair.

2.2 Bandwidth Isolation Guarantee in SDN-NV

Here, we conduct experiments to demonstrate the overloaded link problem (of Fig. 1a) in SDN-NV. For experiments, we use Libera, the latest open-source NH. Also, we set the physical network as a 4-ary fat-tree topology (Fig. 3a) and create one to four tenants (from tenants A to D). Each tenant has one entity pair and creates a custom topology, which is composed of 10 virtual switches (Fig. 3b). The link capacity of the physical network is set to 2 Gbps so that a single path can support up to 2 Gbps. In

4. A path consists of links between switches.

TABLE 1
Related Studies Comparison—SDN-NV

Focus aspect		FlowVisor [4]	FlowN [5]	OpenVirteX [6]	CoVisor [7]	Libera [9]	V-Sight [32]	TeaVisor
		Initial NH	Scalable NH	Improved address virtualization	Flow rule merge	SDN-NV in datacenter	Network monitoring	Bandwidth isolation guarantee
Virtualization	Topology	△ (limited)	○	○	○	○	○	○
	Address	✗	○	○	✗	○	○	○
	Path	✗	✗	✗	✗	✗	✗	○
Bandwidth isolation guarantee		✗	✗	✗	✗	✗	✗	○

addition, the physical network of the 4-ary fat-tree topology supports up to four paths for each entity pair.

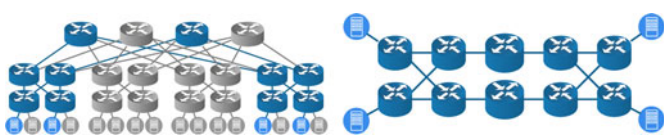
In the experiments, each entity pair requests 2 Gbps, which is the capacity of a single path. To show the worst case of the overloaded link problem, we allow the packets of all entity pairs to traverse through an identical physical network path. This is a plausible scenario because tenants calculate paths without knowledge of any other tenants. Each tenant generates traffic sequentially at 15 s intervals. Thus, 15 s after tenant A generates traffic, tenant B generates traffic. This is to see the interference between tenants.

Fig. 4 shows the network throughput of the tenants. The x-axis corresponds to each tenant, and the y-axis is the network throughput shown as box plots. As shown in Fig. 4a, with tenant A alone, it stably achieves 1,950 Mbps on average, which is close to its requirement. However, when the number of tenants increases to four (Fig. 4d), the throughput of tenant A varies significantly, from 466 to 1,916 Mbps. The throughput of tenant A fluctuates by up to 4.1×, which means that its bandwidth is not isolated at all.

Aside from the fluctuation, the worst throughput of each tenant, which is at the bottom of the box plot, indicates a serious problem. When the number of tenants is two (Fig. 4b), the worst throughput of the two tenants (i.e., A and B) is 820 Mbps on average. The worst throughput of the three tenants (Fig. 4c) and four tenants (Fig. 4d) are 633 Mbps and 460 Mbps, respectively. The results show that no tenant is sufficiently allocated the required bandwidth. As explained in the previous paragraph, all four tenants use the same path on the physical network, so the links of the path are overloaded. Thus, existing NHs fail to provide bandwidth isolation due to the overloaded link problem.

2.3 Bandwidth Isolation Guarantee and Multipath Routing

We review related works to TeaVisor in two categories: bandwidth isolation guarantee and multipath routing.



(a) Physical network (gray) and a mapped VN (blue). (b) VN topology layout.

Fig. 3. Network topology.

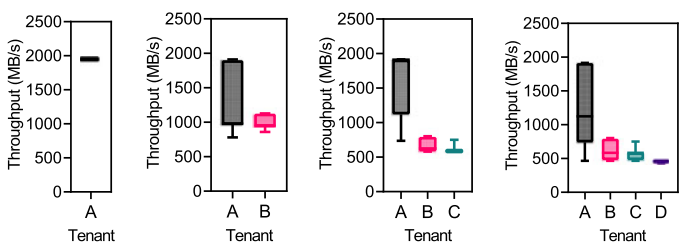
2.3.1 Bandwidth Isolation Guarantee

Multiple tenants in clouds share the datacenter network resources while each runs a different application. Because each tenant and application require different network bandwidths, it is challenging to achieve the bandwidth isolation guarantee of all tenants. Therefore, there are related works that address the bandwidth isolation guarantee of multiple tenants.

Existing studies of bandwidth isolation guarantees can be categorized into two types. The first type refers to studies with the hose model [19], [20], [21], [22], [35], which use proportional sharing [19], rate limiting on the path [20], [22], [35], or switch queueing [21]. The second type refers to studies without the hose model, which apply proportional sharing [19] or rate limiting [16], [23], [24]. In addition, previous studies have attempted to achieve the bandwidth isolation guarantee at different granularities of bandwidth isolation (e.g., per-tenant, per-entity, per-entity pair, per-virtual router, or per-virtual NIC). However, none of these methods resolves the overloaded link problem illustrated in Fig. 1a (Section 1) because SDN-NV allows multiple tenants' SDN controllers to perform their own routing without considering other VNs or the physical network. These existing studies are summarized in Table 2.

2.3.2 Multipath Routing

Extant multipath routing [27], [28], [29], [30] methods focus on improving network utilization by splitting traffic along multiple paths. The previous studies established traffic splitting through a central controller [27], [38], each network hop [28], [30], [39], or software switch [29]. Also, they use global network information [27], congestion information [28], [29], and local/partial information [29], [30]. Although



(a) One tenant. (b) Two tenants. (c) Three tenants. (d) Four tenants

Fig. 4. Lack of bandwidth isolation guarantee in SDN-NV.

TABLE 2
Related Studies Comparison—Bandwidth Isolation Guarantee

Study	VN Model	Bandwidth isolation guarantee	Bandwidth isolation unit	Method	Overloaded link problem
FairCloud PS-P [19]	Hose model	Yes	Per-entity	Queueing in tree topology	Not solved
ElasticSwitch [20]	Hose model	Yes	Per-entity pair	Rate allocation, limiting	
Trinity [21]	Hose model	Yes	Per-entity pair	Switch priority queueing	
eBA [22]	Hose model	Yes	Per-entity pair	Rate control algorithm	
PicNIC [35]	Hose mode	Yes	Per-virtual NIC	Weight-based rate limiting	
FairCloud PS-L/N [19]	NA	No	Per-tenant or -entity	Proportional share	
Seawall [16]	NA	No	Per-source entity	Weight-based rate limiting	
NetShare [23]	NA	No	Per-tenant	Weight-based rate limiting	
CreditBank [24]	Virtual router model	No	Per-virtual router	Credit-based rate limiting	
TeaVisor	SDN-NV	Yes	Per-entity pair	Multipath routing	Solved

they are successful in improving network utilization, these studies do not attain the bandwidth isolation guarantee because these studies are basically routing schemes. In SDN-NV, the NH allows SDN controllers of tenants to perform routing. In other words, each tenant runs its own routing scheme without knowing the existence of other tenants and traffic. Thus, SDN-NV cannot use the previous multipath routing scheme because the routing of tenants (path establishment from SDN controllers) and multipath routing cannot work together for bandwidth isolation. TeaVisor solves this problem through path virtualization.

3 TeaVisor Design

3.1 Workflow

Fig. 5 shows the architecture of TeaVisor. TeaVisor enables each tenant to specify the bandwidth requirements for its end entities (❶-1). So, bandwidth requirements are entered per entity pair. TeaVisor supports two types of bandwidth requirements: minimum (*Min*) and maximum (*Max*). Also, TeaVisor collects physical network information (e.g., total and available link capacities) through network monitoring (❶-2).

The workflow of TeaVisor is as follows. TeaVisor performs bandwidth isolation guarantee per flow of entity pair. When the first packet of a flow between an entity pair arrives, the physical network notifies TeaVisor of the new traffic as an event (❷-1), and the event is delivered to an SDN controller (❷-2). Then, the SDN controller performs routing between the entity pair, resulting in a set of flow rules for the switches, and the flow rules are issued to TeaVisor (❸). Upon receiving the flow rules, path virtualization (Section 3.2) crafts a virtual path (vPath) (❹). The vPath is an end-to-end forwarding path for the entity pair consisting of a number of virtual switches and links.

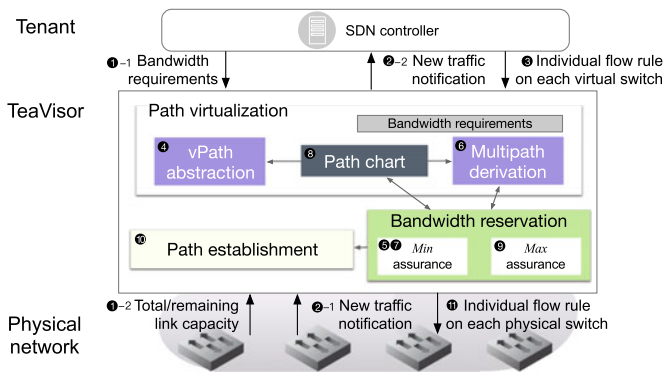


Fig. 5. TeaVisor architecture and workflow.

Then, bandwidth reservation component (Section 3.3) fulfills the bandwidth isolation guarantee (❺) by utilizing multiple physical paths (pPaths) and reserving bandwidth's *Min* requirements on switches and links that comprise the pPath. This component works with path virtualization component (❻–❼). Note that TeaVisor maintains the mappings of vPath and pPaths as path charts (Section 3.2.3) for tenant network management (❽). After reserving *Min*, bandwidth reservation component periodically checks the idle network capacity and reserves additional bandwidth for work conserving up to *Max* (❾). All paths and the reserved bandwidth are realized in the physical network via path establishment component (❿–⓫). A key aspect of TeaVisor is that it satisfies both the *Min* and *Max* of a vPath via multiple pPaths. We use the expression "satisfying *Min* on a vPath or an entity pair" to mean "reserving the *Min* amount for pPaths transmitting packets for the entity pair of the vPath."

3.2 Path Virtualization

Path virtualization supports multipath routing by 1) crafting the vPath for an entity pair from flow rules and 2) identifying pPaths for the vPath. Because NH is separate from tenant controllers, it can receive only flow rules from tenant controllers, which makes it difficult to obtain the end-to-end path. This is a fundamental challenge of any existing NH (e.g., OpenVirteX [6], and Libera [9]) in SDN-NV. Note that the end-to-end path is essential for multipath routing. To address this challenge, we propose "vPath abstraction" and "rule chaining" in TeaVisor which is to enable the construction of the end-to-end path.

3.2.1 vPath Abstraction

Fig. 6 illustrates the process of vPath abstraction. From flow rules obtained from the SDN controllers, TeaVisor connects them from start to finish (i.e., source entity to destination entity) via rule chaining (explained below). Specifically, when a new flow rule arrives, TeaVisor groups the flow rule according to the path to which it belongs. Path

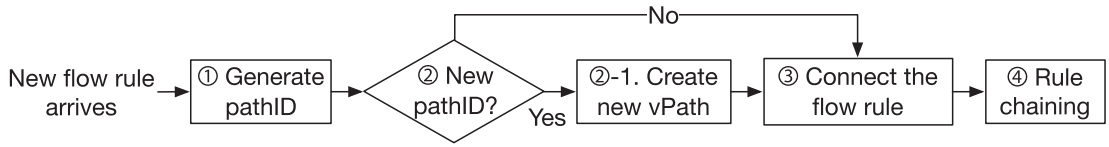


Fig. 6. vPath abstraction process.

virtualization generates a "pathID," an ID for each vPath from the flow rule (① of Fig. 6). PathIDs are used to identify both vPaths and entity pairs. The pathID is a combination of the tenant identifier and IP addresses of the source and destination entities. TeaVisor checks whether a pathID is new, and if so, it indicates that the flow rule of a new vPath has arrived (②). Then, TeaVisor prepares a new vPath object to be created (②-1). Subsequently, whenever a new flow rule arrives, TeaVisor checks whether the vPath of its pathID already exists (③).

If it does, the flow rule is used for rule chaining (④) as follows. Fig. 7 presents an example of rule chaining. Because each flow rule of the switch is intended for packet forwarding, it has in- and out-port information within a switch. For example, for switch 1 in Fig. 7, the in-port is A, and the out-port is B. The out-port is connected to the in-port of another switch. In this case, out-port B is connected to port C of switch 2. In this manner, TeaVisor traces flow rules one at a time, starting from the source to the destination entities. When the end-to-end path is identified, the vPath abstraction is completed.

3.2.2 Multipath Derivation

Based on the vPath, multipath derivation identifies pPaths, a set of physical switches and links to deliver the packets of entity pairs. The first pPath is derived from the vPath as a set of physical switches and links mapped to the virtual switches and virtual links of the vPath, and it is denoted as the "main path" (mPath). For example, in Fig. 2, mPath is derived from the vPath of physical switches (e.g., pS1 to pS4) and the link that connects them. However, because the mPath is determined based on the vPath, the links of mPath may become overloaded, as shown in Fig. 1a. To address this issue, TeaVisor uses multipath routing, in which additional pPaths are derived for the vPath to satisfy the bandwidth requirements. We call the additional paths for satisfying the bandwidth requirement as "extra paths" (exPaths). With this, the traffic can be transmitted over multiple pPaths.

To utilize exPaths, multipath derivation designs a cost-based routing algorithm. The cost of the algorithm is the remaining bandwidth per link, so the algorithm calculates the path having the highest remaining bandwidth in the physical network. Additionally, the routing algorithm identifies exPaths that are disjointed from the mPath of the same entity pair. Finally, bandwidth reservation component (Section 3.3) achieves the bandwidth isolation guarantee by utilizing the exPaths derived by this routing algorithm.

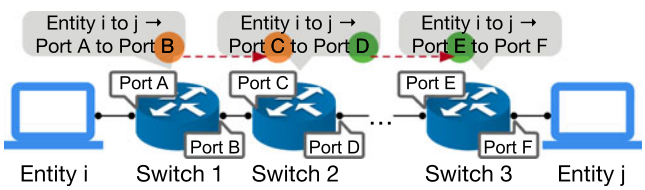


Fig. 7. Rule chaining example.

3.2.3 Path Chart

The purpose of path chart is to support the network management of SDN controllers (e.g., network monitoring) in the presence of the vPath and pPaths. Path chart maintains the mappings of flow rules in the vPath and pPaths. With such mappings, whenever the SDN controller needs to manage its VN, path chart determines the corresponding pPath flow rules. We use the mapping structures for flow rules proposed in [40]. If the tenant modifies the flow rule of a vPath, the flow rules in the physical network mapped to the modified flow rule are also modified accordingly. In addition, path chart provides statistics of all physical flow rules mapped to the requested flow rule so that tenants can monitor the statistics of the flow rules. TeaVisor uses the algorithms proposed by [32] so that it delivers the aggregated values of multiple statistical values of pPaths as the statistics of a single vPath.

3.3 Bandwidth Reservation

The two bandwidth requirements, *Min* and Max, mean the following for a vPath: *Min* is the amount of bandwidth that is always to be provided for the vPath, and up to *Max* bandwidth can be reserved on the vPath in a best-effort manner for work conserving. Tenants can request the following bandwidth requirement combinations: 1) *Min* without *Max* (denoted [*Min*, *Min*]) and 2) *Min* with *Max* ([*Min*, *Max*]). By setting infinite (*Inf*) as the *Max* value, tenants can request that datacenter can allocate all additional bandwidth to the entity pair without limitations. Note that the optimal solution to the bandwidth reservation problem in NV or SDN-NV is NP-hard [41]; thus, we design two greedy heuristics (*Min* assurance and *Max* assurance). Also, in datacenters, the number of tenants, end entities, and the traffic between the end entities are dynamically generated, so we perform bandwidth reservation as the traffic of each end entity is generated. If the *Min* and *Max* of an entity pair change, bandwidth reservation reruns for the changed values.

Fig. 8 illustrates *Min* and *Max* assurance scenarios. We assume two entity pairs, Pair1, whose mPath is A-B-C-D and Pair2 with A-G-D as its mPath. Both have bandwidth requirements of [800 Mbps, 1,200 Mbps], and the physical network has 1 Gbps links. As the traffic of each entity pair is generated, *Min* assurance satisfies *Min*. For example, for Pair1, 300 Mbps is reserved on mPath (A-B-C-D), and the other 500 Mbps is reserved on exPath (A-G-D). For Pair2, 500 Mbps is reserved on mPath (A-G-D) and the other 300 Mbps on exPath for Pair2 (A-B-C-D). If the *Min* of each pair is satisfied (*Min* satisfaction), *Max* assurance runs

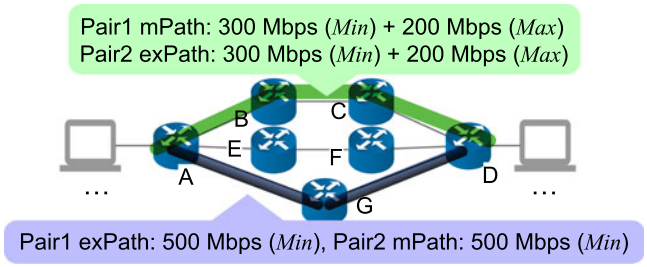


Fig. 8. Bandwidth reservation example.

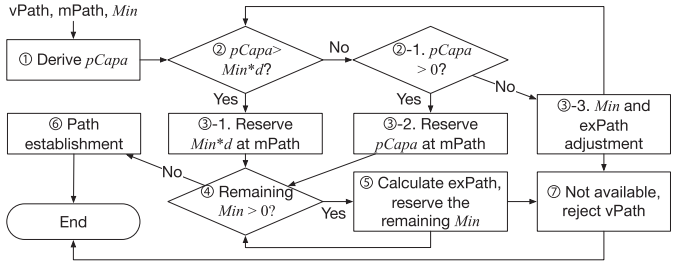


Fig. 9. *Min* assurance.

periodically to reserve additional bandwidth using the idle capacity of the physical network to both pPaths (to be explained in Section 3.3.2). In the example of Fig. 8, *Max* assurance can get 200 Mbps more for the mPath and exPath each. We explain *Min* assurance and *Max* assurance in detail next.

3.3.1 Min Assurance

Min assurance achieves the bandwidth isolation guarantee using multipath routing. The idea of *Min* assurance is to satisfy as many entity pairs as the physical network capacity permits by properly utilizing mPaths and exPaths. Also, we ensure that the mPath is preserved in the physical network and that the packets flow through the mPath because it is the one directly derived from the vPath from tenants. However, there can be cases wherein the mPath of a new entity pair may not be able to transmit packets because other entity pairs have already consumed all of the bandwidth of the links in the mPath. For example, in Fig. 8, assume that a new entity pair initiates its traffic, its bandwidth requirement is [800 Mbps, 1200 Mbps], and the mPath for the entity pair is A-B-C-D. In this situation, *Min* assurance cannot reserve bandwidth on the mPath because the other entity pairs have already reserved it all. As a remedy to this situation, *Min* assurance secures a certain amount of bandwidth for an entity pair through parameter d (explained later).

Min assurance works with a vPath (from vPath abstraction), the corresponding mPath (from multipath derivation), and the *Min* for the entity pair. *Min* assurance uses two parameters: physical capacity (*pCapa*) and bandwidth reservation ratio (d). *pCapa* is the amount of bandwidth that can be reserved for a pPath. The amount of available bandwidth of a pPath is determined by the most congested link of a pPath; so, *pCapa* is the remaining bandwidth of the most congested link. The tunable parameter, d, determines the portion of *Min* to be reserved in the mPath, ranging from zero to one. So, the bandwidth for the mPath is *Min***d*.

First, *Min* assurance calculates the *pCapa* of the given mPath (① of Fig. 9). *Min* assurance then checks that the *pCapa* of mPath is sufficient for *Min***d* (②). If so, *pCapa* satisfies *Min***d* (*pCapa* > *Min***d*), and *Min***d* is reserved on the mPath (③-1). Here, if d is 1, the entire amount of *Min* can be reserved for mPath. On the other hand, if d approaches zero, the reserved bandwidth for mPath becomes smaller. By reserving *Min***d*, the remaining *Min* becomes *Min**(1 - d). Note that, from here, *Min* is updated as the remaining *Min*.

Next, if *Min* > 0 ("yes" condition of ④), *Min* assurance utilizes the exPaths and reserves the *Min* on the exPaths (⑤). Specifically, exPaths are created one at a time with the assistance of multipath derivation. If the *pCapa* of the created exPath ($pCapa^{exPath}$) is sufficient for *Min* ($pCapa^{exPath} > Min$), *Min* assurance reserves *Min* on exPath and completes the exPath creation. However, if $pCapa^{exPath} < Min$, *Min* assurance reserves $pCapa^{exPath}$ on the exPath, calculating the remaining *Min* (*Min* - $pCapa^{exPath}$) and creating another exPath to reserve the remaining *Min*. Routine ⑤ is repeated until the remaining *Min* reaches zero. When repeating this routine, if no additional exPaths can be created and *Min* is still left over (*Min* > 0), it means that the network's capacity is deficient. Thus, bandwidth reservation for the entity pair is rejected (⑦). On the contrary, if *Min* = 0, meaning that all of *Min* is reserved ("no" condition of ④), *Min* assurance is completed, and "path establishment" is called to install the created paths and bandwidth reservations (⑥).

On the other hand, if *pCapa* is less than *Min***d* from ②, *Min* assurance first checks whether the *pCapa* of mPath is zero (②-1). If not, we only reserve the amount of *pCapa* on mPath (③-2), and the remaining *Min* becomes *Min* - *pCapa*.[5] We then reserve the remaining *Min* on exPaths using the process starting from ④.

If *pCapa* is zero, no bandwidth can be reserved on the mPath because the links are already fully utilized. In this situation, *Min* assurance checks whether additional *pCapa* can be secured for the mPath by adjusting the bandwidth reserved for other entity pairs' *Max* and exPaths (③-3) as follows.

First, TeaVisor considers the bandwidth reserved for other entity pairs' *Max* (*Max* adjustment, ③-3). Because the bandwidth for *Max* is not mandatory, it can be dynamically adjusted. Assume that, in Fig. 8, a tenant creates a new mPath of A-B-C-D (Pair3) whose *pCapa* = 0. In this situation, *Max* adjustment can secure bandwidth for a new entity pair's mPath by taking bandwidth reserved for *Max* (e.g., 200 Mbps of Pair1 and 200 Mbps of Pair2). Thus, if the *Max* adjustment can spare some bandwidth, TeaVisor uses the bandwidth for the mPath.

When the *Max* adjustment cannot spare any bandwidth, TeaVisor then checks whether the other entity pairs' exPaths can be adjusted (exPath adjustment, ③-3). exPath adjustment checks whether the new entity pair's mPath has any overlap with others' exPaths. For example, suppose a new entity (i.e., Pair3) installs its mPath as A-B-C-D (Fig. 8).

5. Note that *Min* assurance tries to allocate *Min***d* to mPath on a best-effort basis. This means that even if only a small amount of bandwidth beyond *Min***d* is available for the mPath, *Min* assurance does not reject the entity pair. This decision is to facilitate the acceptance of as many entity pairs as possible.

Also, assume that all the bandwidth reserved for *Max* is already in use for other mPaths. Then, *Min* assurance for the new entity pair considers whether the existing entity pairs' exPaths overlap with Pair3's mPath (A-B-C-D). In our example, Pair2's exPath overlaps with the newly requested Pair3's mPath (A-B-C-D). However, the bandwidth reserved for the exPaths is to assure *Min*. Thus, we cannot simply reduce their bandwidth. Instead, *Min* assurance finds another pPath that can replace the existing exPath here. In our example, luckily, another path, A-E-F-D, can deliver the packets for Pair2. Then, *Min* assurance for Pair3 secures the bandwidth by replacing the existing exPath (e.g., A-B-C-D of Pair2) with A-E-F-D. So Pair3's mPath becomes A-B-C-D.

Algorithm 1. *Max* Assurance

- Min^a, Max^a: *Min*, *Max* of the entity pair that pPath a belongs to
- $R.Max^a$: Amount of additional bandwidth reserved for *Max* on the entity pair to which pPath a belongs
- u_k: The current throughput of pPath k
- $l_k.remn$: remaining bandwidth of l_k
- $B(i,j)$, $M(i.j)$: Select bigger and smaller values, respectively

1: **for** l_i in physical network links **do**
2: **if** remaining bandwidth of $l_i > 0$ **then**
3: Find P_n, a set of pPaths whose *pCapa*s are determined by l_i
4: **for** k in P_n **do**
5: $r^k = B(0, Max^k - Min^k - R.Max^{kP})$
6: **if** ES **then** Resv = $M(l_i/|P_n|, r^k)$
7: **if** PBR **then**
8: Resv = $M(l_i * Min^k / \sum_{i \in P_n} Min^i, r^k)$
9: **if** PNU **then**
10: Resv = $M(l_i * u_k / \sum_{i \in P_n} u_i, r^k)$
11: $R.Max^k += Resv$
12: For each link l_k belong to k, $l_k.remn -= Resv$
13: Request increasing the rate of the k

After securing bandwidth for the mPath from the other entity pairs' *Max* or exPath, *Min* assurance returns to ②. However, if *pCapa* cannot be secured, even after *Max* and exPath adjustments, the physical network cannot accept the entity pair with *Min*. In this case, TeaVisor must reject the entity pair's vPath (⑦).

The *Min* assurance workflow has an important parameter, d. We evaluate the impact of d on the bandwidth isolation guarantee. We also evaluate the efficacy of exPath adjustment in Section 4.4.

3.3.2 Max Assurance

Max assurance provides work conserving for TeaVisor. Note that it does not create or remove paths; instead, it only adds bandwidth to existing pPaths. Algorithm 1 describes how *Max* assurance works. *Max* assurance runs at regular intervals to check whether there is any available (idle) bandwidth within the physical links (line 2 in Algorithm 1). Note that because the remaining link capacity that can be used for *Max* dynamically changes, periodically monitoring the remaining bandwidth is necessary for *Max* assurance. If a physical link (l_i) has idle bandwidth, Algorithm 1 finds pPaths (P_n) whose *pCapa* is decided by l_i (line 3). Then, according to the three policies below, *Max* assurance divides the idle bandwidth of l_i for P_n into three policies:

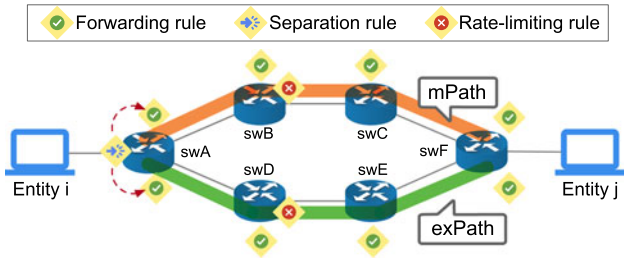


Fig. 10. Path establishment example.

- Equal share (ES, line 6): divide the idle bandwidth equally between P_n.
- Proportional share for bandwidth requirements (PBR, line 7): divide the idle bandwidth according to the ratio of *Min* between P_n.
- Proportional share for actual network usage (PNU, line 9): divide idle bandwidth based on the ratio of the monitoring results on the pPaths of P_n (actual throughput used within the reserved bandwidth).

For all three policies, *Max* assurance ensures that the total bandwidth reserved for an entity pair is lower than *Max* (lines 5–10).

3.4 Path Establishment

Using path virtualization (Section 3.2) and bandwidth reservation (Section 3.3), TeaVisor derives multiple pPaths (i.e., mPath and exPaths) and allocates the reserved bandwidth to them. Path establishment installs flow rules to substantiate bandwidth isolation in the physical network. Specifically, pPaths and their reserved bandwidth are substantiated via three types of flow rules: 1) forwarding rule, 2) separation rule, and 3) rate-limiting rule.

The forwarding rule delivers packets from the in-port to the out-port within a switch. For mPath, the forwarding rules are installed by flow rules sent by SDN controllers of tenants, which includes the translation between the virtual network and physical network (i.e., host address, switch address, in-port, and out-port). TeaVisor also creates forwarding rules for exPaths to enable packet forwarding through both mPaths and exPaths.

In addition to the forwarding rules, TeaVisor installs two additional rules: 1) separation rule and 2) rate-limiting rules. The separation rule enables multipath routing to divide packets into multiple pPaths. So, TeaVisor installs a separation rule per entity pair to divide the packets. Suppose that an entity pair has two pPaths: mPath of swA-swB-swC-swF and the exPath of swA-swD-swE-swF, as shown in Fig. 10. Then, the separation rule is installed at swA, where the two paths are divided. The rate-limiting rule is to limit the packet transmission rate conforming to the reserved bandwidth and to prevent excessive consumption of bandwidth. Because both mPath and exPaths have their reserved bandwidth (Section 3.3.1), the rate-limiting rule is installed for both the mPath and exPaths. In the example of Fig. 10, TeaVisor installs the rate-limiting rules for mPath and exPath on swB and swD as the separation rule is installed in the swA. The rate-limiting rule can also be installed at swA. However, it is known that each

switch has a limited memory to store rules. So, we install the rate-limiting rule at swB and swD to avoid installing too many rules in swA.

4 Implementation and Evaluation

We implement all the components of TeaVisor based on Libera (4.7K LoC of Java). We use OpenFlow 1.3, the de facto interface in SDN systems commonly supported by commercial SDN switches. TeaVisor implements the separation and rate-limiting rules using the group table and meter table mechanisms of OpenFlow 1.3. We empirically set the value of parameter d to 0.5 and perform network monitoring for *Max* assurance at every 5 s. We compare the performance of TeaVisor against two NHs:

- ***Libera.*** This is the latest open-source NH without any bandwidth isolation guarantee. We measure the improvement of TeaVisor over Libera.
- ***Libera+RL.*** On top of Libera, we implement the rate-limiting algorithm (similar to [16], [20], [22]). This is to simulate bandwidth isolation by applying rate-limiting to each path. The purpose is to show that the existing studies are insufficient for handling the overloaded link problem.

4.1 Evaluation Environments

Three servers of Intel Xeon E5-2600 CPUs are used for our evaluations. The servers are connected through a 10 GbE switch. We run a Mininet of Open vSwitches [42] for the physical network, and ONOS [43] is used as an SDN controller. The network topology is configured similarly to that in Section 2.2. For example, the physical network is set as a 4-ary fat-tree topology which is commonly used in datacenters (Fig. 3a). Also, the VN topology is set as shown in Fig. 3b, consisting of ten switches and four entities. In evaluation, a tenant has four entity pairs, and each entity pair sends 128 TCP connections using iperf3 [44] at their full speed.

In our experiments, we set the entity pairs to send packets higher than the *Min* to show its isolation guarantee. However, it is possible that the actual bandwidth can go below *Min* due to TCP behavior. Yet, from the bandwidth isolation point of view, the isolation is not broken with a traffic that sends less than *Min*. So, TeaVisor does not need to take any action on such traffic. What TeaVisor is designed to act is when the entity pairs request a high bandwidth and send a high volume of traffic; in that case, the traffic of the entity pairs interfere with each other. This is when TeaVisor kicks in for the bandwidth isolation.

The bandwidth requirements are entered per entity pair. Note that *Max* is for work-conserving; thus, it is only meaningful after *Min* is guaranteed. So we do not consider the case in which only *Max* is required without *Min*, which is an identical assumption in other existing bandwidth guarantee studies [15], [31]. We experiment with four different bandwidth requirement types for entity pairs as follows:

- ***[Min, Min]:*** All entity pairs only request *Min*, so no additional bandwidth exceeding *Min* is allowed.
- ***[Min, Max]:*** All entity pairs request both *Min* and *Max*. For *Max*, we empirically set *Max* as 120% of *Min* because it is more challenging to guarantee bandwidth isolation when the gap between *Max* and *Min* is small.
- ***[Min, Inf]:*** All entity pairs request *Min* and *Max* that *Max* is set as *Inf*, which means that each entity pair can receive an unlimited amount of additional bandwidth on top of *Min*.
- ***Hybrid***: The above three types are combined in the requests. Specifically, in the experiment of the hybrid type in Fig. 15d, entity pairs 1, 4, 9, 11 and 14 (of the x-axis) request [*Min, Min*], pairs 2, 3, 8, 10 and 13 request [*Min, Max*], and pairs 5, 6, 7, 12, 15, and 16 request [*Min, Inf*].

Our evaluation consists of four sets of experiments: 1) bandwidth isolation guarantee, 2) effects of *Max* assurance policies, 3) effects of parameter d and exPath adjustment, and 4) overheads of TeaVisor. We explain the setting of each experiment in detail in the following subsections.

4.1.1 Bandwidth Isolation Guarantee

We vary the total amount of *Min* (total *Min*) values (sum of the *Min* of all the pairs) proportional to the entire network capacity (i.e., 37.5% to 100% because we find that the results under 37.5% are similar to 37.5%) with 16 entities of four tenants. Although the physical network of experiments has a symmetric topology (4-ary fat tree), we evaluate TeaVisor with asymmetric scenarios in which the bandwidths for the entity pairs differ between links. The asymmetry is configured by setting different *Min*s for entity pairs so that the pairs consume different bandwidths. The total *Min* is distributed among the 16 entity pairs as follows. Half of the entity pairs (eight pairs) account for 90% of the total *Min*, and each entity pair requests the same *Min*, 11.25% of the total *Min* (90% of the total *Min* divided by eight). The remaining eight entity pairs account for the remaining 10% of the total *Min*, and each pair requests 1.25% of the total *Min* (10% of the total *Min* divided by eight). For example, in Fig. 15a, entity pairs 1, 4, 5, 8, 9, 12, 13, and 16 (of the x-axis) request 90%, and pairs 2, 3, 6, 7, 10, 11, 14, and 15 request the remaining 10% of the total *Min*.

Also, we change the number of tenants from one to four. The bandwidth requirement of one tenant is 25% of the physical network capacity. Thus, as the tenants increase to four, the number of entity pairs and the bandwidth requirement increase proportionally, up to 16 pairs and 100%, respectively.

4.1.2 Effects of Max Assurance Policies

We analyze the effect of the three *Max* assurance policies (i.e., ES, PBR, and PNU) by measuring the additional bandwidth reserved for the active entity pairs (Section 4.3). We present the results obtained with four tenants (16 entity pairs) having a total *Min* of 50% of the physical network capacity, and *Max* set as *Inf* ([*Min*, *Inf*] type).

4.1.3 Effects of Parameter d and exPath Adjustment

We evaluate the effects of parameter d and exPath adjustment used in *Min* assurance (Section 4.4). We have also

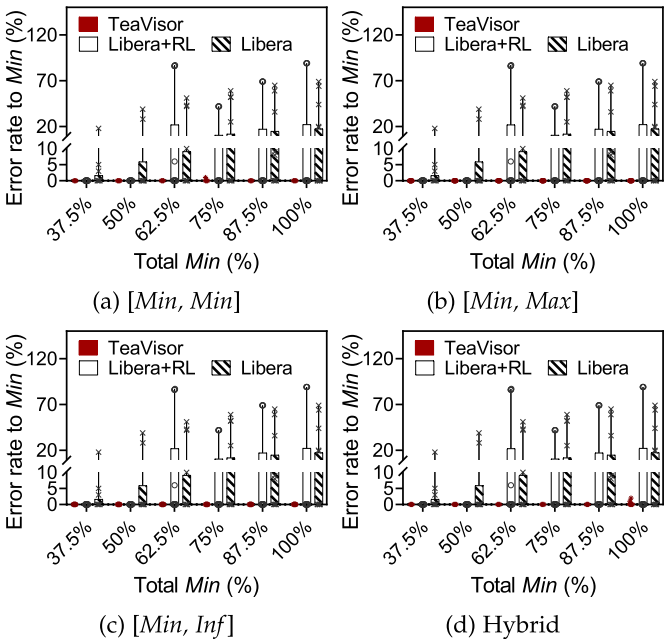


Fig. 11. Error rates of *Min* when varying the total *Min*.

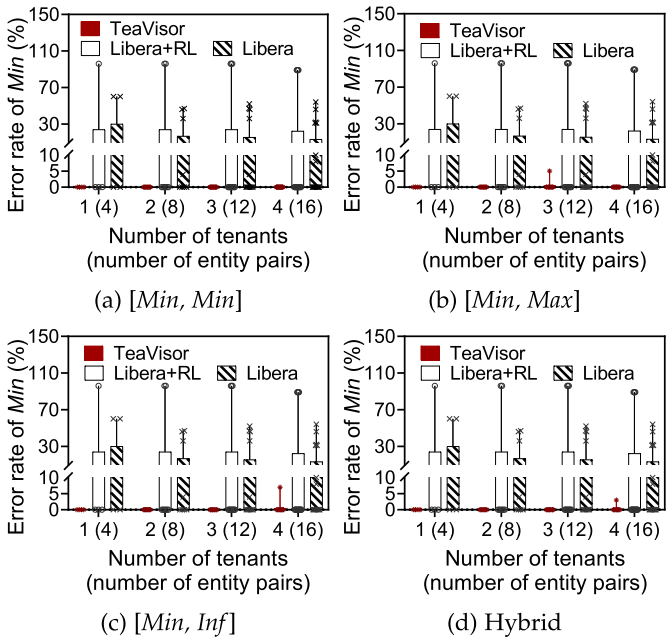


Fig. 12. Error rates of *Min* when varying the number of tenants.

measured the error rates when the *Max* adjustment is on or off. However, the error rates with the *Max* adjustment reduce a similar amount of *Max* (e.g., 20% additional bandwidth by *Max* in [*Min*, *Max*] types). Thus, we only consider exPath adjustment here. We vary d between 0.1 and 0.9 and measure the error rates of *Min*. In addition, to analyze the effect of the exPath adjustment, we compare the error rates of *Min* when the exPath adjustment is either on or off. This evaluation is conducted with four tenants and 16 entity pairs.

4.1.4 Overheads

We evaluate the overheads of TeaVisor from two aspects: control traffic consumption and memory consumption of TeaVisor. Control traffic reflects the overhead of path virtualization and bandwidth reservation components because the multiple pPaths and reserved bandwidth require additional flow rules. Also, because the TeaVisor's components are parts of NH, the memory consumption shows the increased resource consumption of TeaVisor. We present the results when the number of tenants changes from one to four.

4.2 Bandwidth Isolation Guarantee

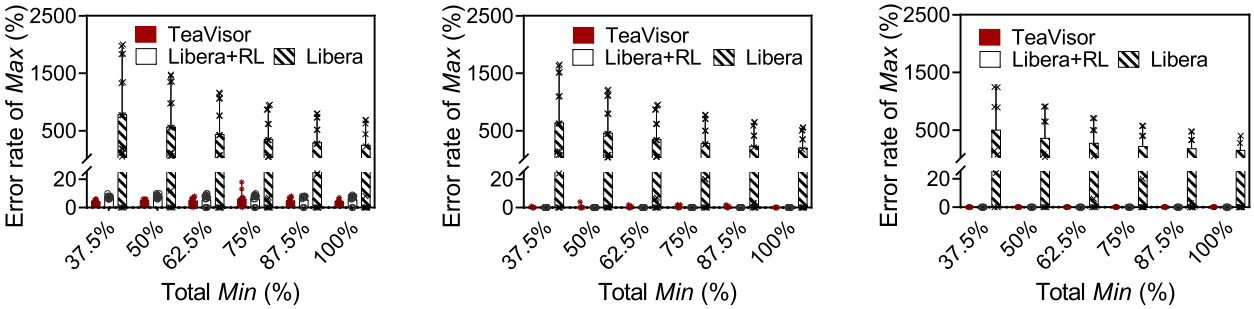
In this subsection, we present the bandwidth isolation guarantee of TeaVisor through the error rates with regard to *Min* (Section 4.2.1) and *Max* (Section 4.2.2). We also show the actual throughput of entity pairs with *Min* and *Max* (Section 4.2.3). In addition, we check how well the bandwidth of the entity pair is isolated, which we call fairness in Section 4.2.4.

4.2.1 Minimum Bandwidth Requirements

Figs. 11 and 12 present the error rates of *Min* measured when varying the total *Min* and the number of tenants, respectively. The y-axes are the error rates of *Min*, which implies the rate of the bandwidth lower than *Min*. Thus, an error rate of 0% means that the entity pair receives a bandwidth of at least *Min* during the experiments. The x-axes of Figs. 11 and 12 represent the total *Min* and the number of tenants, respectively. Both figures include four subfigures corresponding to the four bandwidth requirement types. The markers, bars, and whiskers in the figures are the individual value, the mean, and the range of the error rates, respectively. Hence, cases where the bar or whisker is barely visible represent entity pairs whose error rates have similar average and range values (e.g., TeaVisor's error rates at 37.5% in Fig. 11a are almost zero, so the bars and whiskers are all marked at zero).

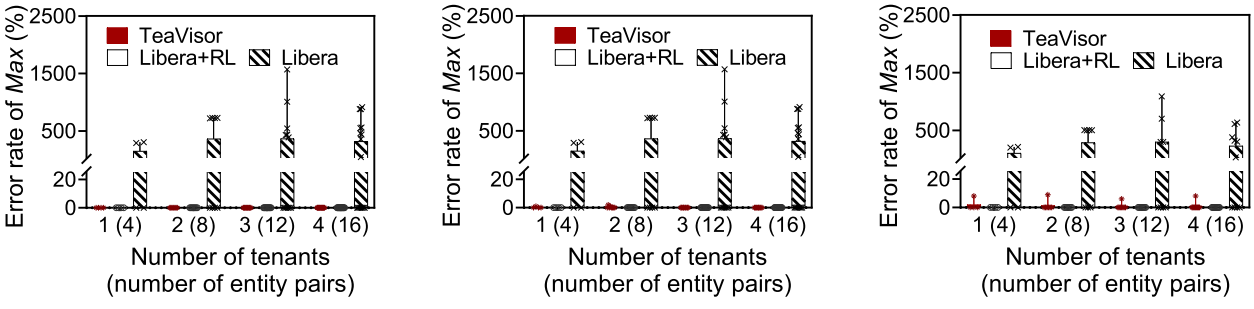
Total Min. When the total *Min* changes, Fig. 11a shows the error rates of *Min* in the [*Min*, *Min*] type. TeaVisor, Libera+RL, and Libera show error rates on average of 0.01%, 12%, and 10%, respectively. From the three NHs, we first compare TeaVisor with Libera+RL. Up to the total *Min* of 50%, both NHs have similar error rates (0% on average). This is because the physical links are not overloaded. However, once the total *Min* exceeds 50%, the network links begin to overload. Because Libera+RL cannot solve the overloaded link problem, its error rates dramatically increase to 89%, whereas TeaVisor maintains 0%.

Second, we compare Libera+RL and Libera in Fig. 11a. Libera+RL's average error rate is a little higher than Libera's. To see the reason, we calculate the peak error rates of Libera+RL and Libera, which are 89% and 69%, respectively. Also, from the 16 active entity pairs, non-zero error rates appear in 4 pairs for Libera+RL and 6 pairs for Libera, respectively. This means that, although Libera+RL satisfies the *Min* more than Libera (by 2 pairs), the error rates of the unsatisfied entity pairs are much higher than Libera, causing a significant increase in the average error rate. This is because of the rate limiting algorithm used by Libera+RL, which leads to some entity pairs getting their requested *Min* bandwidth, while the rest suffer from the bandwidth starvation, which leads to high error rates (here, 89%). On the



(a) [*Min*, *Min*] (b) [*Min*, *Max*] (c) Hybrid

Fig. 13. Error rates of *Max* when varying the total *Min*.



(a) [*Min*, *Min*] (b) [*Min*, *Max*] (c) Hybrid

Fig. 14. Error rates of *Max* when varying the number of tenants.

other hand, Libera does not control any bandwidth, so the traffic between entity pairs competes for the limited bandwidth. Thus, although many of them do not satisfy *Min* requirements, none of them suffers from serious starvation. Therefore, compared to Libera+RL, Libera results in lower peak error rates.

Figs. 11b, 11c, and 11d show the error rates of *Min* for [*Min*, *Max*], [*Min*, *Inf*] and hybrid types. We observe that TeaVisor achieves near 0% error rates (0.01% on average) for all experiments of the three figures, demonstrating that it successfully guarantees *Min* even for the [*Min*, *Max*], [*Min*, *Inf*] and hybrid types wherein the additional bandwidth is given to the entity pairs. On the other hand, Libera+RL and Libera show high error rates: the error rates of the two NHs are up to 89% for [*Min*, *Max*] and 69% for [*Min*, *Inf*] and hybrid types. In summary, TeaVisor entirely decreases the error rates seen with the existing NHs by 99.9%, representing an improvement of 5633× over Libera+RL and Libera.

Number of Tenants. Fig. 12a shows the error rates of *Min* when the number of tenants increases. In Fig. 12a of [*Min*, *Min*] type, the peak error rates of Libera+RL and Libera, even for only one tenant, are 96% and 60%, respectively. We observe similar results with two, three, and four tenants. On the other hand, TeaVisor achieves an average of 0% error rates for all number of tenants. The results demonstrate that TeaVisor successfully guarantees *Min* even after varying the number of tenants. Note that these results are due to the same reason explained for the changes in total *Min*.

In addition, the results in Figs. 12b, 12c and 12d show the error rates of *Min* for [*Min*, *Max*], [*Min*, *Inf*] and hybrid types. The three figures show similar trends as those in Fig. 12a. For example, Libera+RL and Libera of the three figures show similar high error rates for all bandwidth requirement types as shown in Fig. 12a, which are up to 96% and 60% on average, respectively. In contrast, TeaVisor always shows near-zero error rates regardless of the number of tenants and the bandwidth requirement types. The only exception case wherein TeaVisor shows error rates higher than 0% is with four tenants in the hybrid case (Fig. 12d), wherein TeaVisor still shows the maximum of 2% errors. However, considering that Libera+RL and Libera exhibit up to 89% and 54% peak errors for that case, the error rates of TeaVisor demonstrate a significant improvement.

4.2.2 *Maximum Bandwidth Requirements*

We evaluate the effect of *Max*. Any amount of bandwidth consumed in excess of *Max* is considered to be an error, so the *Max* error rate of zero indicates that the entity pair is reserved its bandwidth lower than *Max*.

Figs. 13 and 14 show the results obtained when varying the total *Min* and the number of tenants, respectively. The y-axes are the error rates of *Max*, and the x-axes are the total *Min* and the number of tenants. Figs. 13 and 14 include three subfigures corresponding to the three bandwidth requirement types: [*Min*, *Min*], [*Min*, *Max*], and hybrid. The results of the [*Min*, *Inf*] type are not presented here due to the following reasons. *Inf* given to *Max* indicates that the unlimited additional bandwidth can potentially be allocated to entity pairs. In other words, there is no threshold on the additional bandwidth for entity pairs, so the errors for the [*Min*, *Inf*] are not defined. For the same reason, when calculating the error rates for the hybrid case, entity pairs of [*Min*, *Inf*] type (i.e., 5, 6, 7, 12, 15, and 16 of Fig. 14c) are excluded.

Total Min. Fig. 13a shows the error rates when changing the total *Min* for [*Min*, *Min*] of 16 entity pairs. TeaVisor, Libera+RL, and Libera show average error rates of 5%, 6%, and 519%, respectively. Similarly, in Fig. 13b for [*Min*, *Max*], the error rates for TeaVisor, Libera+RL, and Libera are 0.3%, 0%, and 360% on average, respectively. Lastly, Fig. 13c shows the error rates of hybrid type. The average error rates of TeaVisor, Libera+RL, and Libera are 0%, 0%, and 280% on average. We analyze these results as follows. First, Libera shows the highest error rates for all bandwidth requirement types. This is because entity pairs consume bandwidth arbitrarily. Second, TeaVisor and Libera+RL show quite similar error rates on *Max*, meaning that both NHs provide bandwidths lower than *Max*. The reason is that both TeaVisor and Libera+RL limit the bandwidth with rate-limiting rules. In particular, Libera+RL almost does not exceed *Max* because there is no mechanism for additional bandwidth allocation for *Max*.

Number of Tenants. Fig. 14 shows the *Max* error rates when varying the number of tenants. From Fig. 14a, the average error rates for TeaVisor, Libera+RL, and Libera are 0%, 0%, and 280%, respectively. Also, from Fig. 14b, their average error rates are 0.1%, 0%, and 323%, respectively. Lastly, from Fig. 14c, the average error rates are 1%, 0%, and 244%, respectively. The reasons for such results are the same as those discussed in relation to the variation of total *Min* (e.g., Fig. 13).

4.2.3 Throughput

We now explain the actual throughput that each entity pair achieves with four tenants (16 entity pairs). The total *Min* of these experiments is 75% of the entire network capacity. When the total *Min* is set to 75%, *Max* is set to 90% (Section 4.1) for the entity pairs of the [*Min*, *Max*] type. Typically, the headroom is up to 10% of the physical link capacity on average [20]; so we choose a total *Min* of 75% for the experiments.

Fig. 15 includes the four bandwidth requirement types of [*Min*, *Min*], [*Min*, *Max*], [*Min*, *Inf*], and hybrid. The x-axes of the figures list the entity pairs, numbered from 1 to 16, and the bars are their throughputs when using either TeaVisor, Libera+RL, or Libera. The x and circle markers indicate the *Min* and *Max* assigned to each entity pair, respectively. Note that for entity pairs with *Inf*, no *Max* markers are drawn and, instead, an asterisk is added to the entity pairs of the x-axis, such as entity pairs of 5, 6, 7, 12, 15, and 16 in Fig. 15d.

Fig. 15a shows the throughputs for the [*Min*, *Min*] type. TeaVisor always satisfies *Min*. On the other hand, Libera +RL exhibits throughputs 42% lower than *Min* for four entity pairs (4, 5, 12, and 13 in Fig. 15a), and Libera also shows throughputs 47% lower than *Min* for four entity pairs (1, 4, 8, and 16 in Fig. 15a). The reasons for such results are similar to those explained in Section 4.2.1.

Fig. 15b ([*Min*, *Max*] type) shows that TeaVisor successfully satisfies *Min* requirements while Libera+RL and Libera fail, similarly to Fig. 15a. In terms of *Max*, in Libera, ten entity pairs show throughputs higher than *Max* (all entity pairs except 1, 4, 8, 9, 12, and 16 in Fig. 15b). The reasons for these results are similar to those explained in Section 4.2.2.

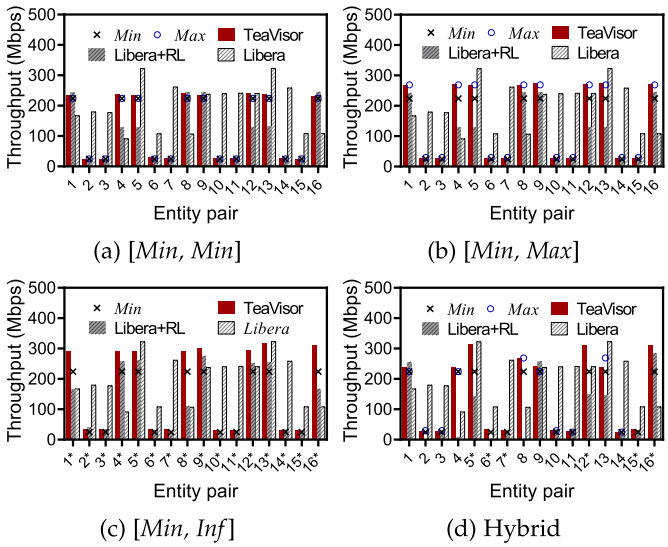
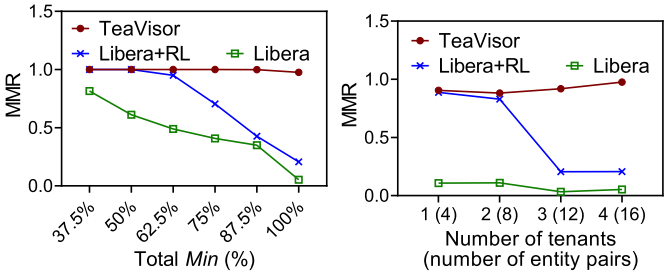


Fig. 15. Throughput gained by individual entities under different bandwidth requirement types.

However, the throughputs of TeaVisor are lower than *Max* for all entity pairs, demonstrating that TeaVisor satisfies the *Max*. Libera+RL also does not exceed the *Max* because it limits the bandwidth of entity pairs to *Min*. However, Libera+RL does not utilize additional bandwidth for work conserving, while TeaVisor reserves available bandwidth within *Max*. Specifically, entity pairs of Libera gain additional throughput of 20% higher than *Min* on average, whereas Libera+RL cannot provide any extra bandwidth.

Fig. 15c shows the results of [*Min*, *Inf*] types. The throughput of TeaVisor is always higher than *Min*, which means that TeaVisor satisfies both *Min* and *Max* (*Inf*). Specifically, the entity pairs show throughputs 34% higher than their *Min* on average. However, for Libera+RL, three entity pairs (1, 8, and 16) show 34% less traffic than their *Min*; and for Libera, four entity pairs (1, 4, 8, and 16) show 47% less throughput than their *Min* on average. These results indicate that both Libera+RL and Libera fail in providing both *Min* and *Max* assurance.

In Fig. 15d, for the hybrid type, we first explain the results with the [*Min*, *Min*] type (i.e., entity pairs 1, 4, 9, 13, and 14). TeaVisor satisfies *Min* for them, but Libera+RL and Libera do not. Specifically, Libera+RL provides 65.5% less throughput than the *Min* for entity pairs 4 and 13. Also, Libera provides the throughput 42% less than *Min* for pairs 1 and 4. Second, for the [*Min*, *Max*] type (i.e., five entity pairs of 2, 3, 8, 10, and 11), TeaVisor satisfies both *Min* and *Max*. For example, for *Min*, the five entity pairs receive higher throughput than *Min*; so, *Min* is satisfied. Also, for *Max*, *Max* for the entities is set as 20% higher than the *Min*, and all the five entity pairs show less throughput than *Max*, showing that *Max* is satisfied. However, Libera+RL and Libera fail in satisfying the requirements. In terms of *Min*, Libera+RL provides 98% less throughput than *Min* for entity pairs 2 and 8 on average. Furthermore, Libera provides 52% less throughput than *Min* for entity pair 8. In terms of *Max*, Libera+RL rarely exceeds *Max* as it limits the throughput of entity pairs by *Min* without considering *Max*. Libera fails in satisfying *Max* because entity pairs 2, 3, 10, and 11 exceed *Max* by 602% on average. Lastly, for the [*Min*, *Inf*] (i.e.,



(a) Varying the total *Min*. (b) Varying the number of tenants.

Fig. 16. Fairness of the *Min* satisfaction.

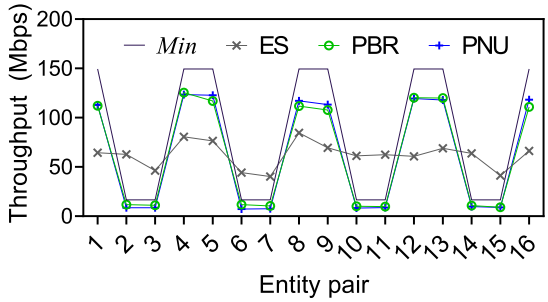


Fig. 17. Changes in throughput for different *Max* assurance policies.

entity pairs 5, 6, 7, 12, 15, and 16), TeaVisor provides 31% higher throughput than *Min* on average for all entity pairs. In contrast, Libera+RL provides 55% less throughput than *Min* for entity pairs 5, 6, and 12, and Libera provides 52% less throughput for entity pair 16. To summarize, TeaVisor guarantees both *Min* and *Max* even when the bandwidth requirements are mixed, while Libera+RL and Libera fail both *Min* and *Max*.

4.2.4 Fairness

We discuss the fairness of the bandwidth isolation guarantee. Here, the fairness means whether TeaVisor can provide a similar degree of bandwidth isolation guarantee to all entity pairs in the network. We measure the fairness by the minimum-maximum ratio (MMR), which is frequently used to measure the fairness of the system performance [45], [46]. In this subsection, we present MMR on *Min*. We have also calculated MMR in terms of *Max* but omit the results, as they are highly similar to those of *Min*.

Specifically, to obtain the MMR, we first calculate the *Min* satisfaction rate, which indicates whether the throughput of the entity pair is satisfied in terms of *Min*. So, the *Min* satisfaction rate is the actual throughput of each entity pair divided by the entity pair's *Min*. The MMR is then calculated by dividing the lowest value among the *Min* satisfaction rates of all entity pairs by the highest value so that the MMR is between 0 and 1. Note that we use satisfaction rates instead of error rates (used in Section 4.2) because TeaVisor typically presents 0% of error rates, which cannot be used as the MMR denominator. MMR 1 means that the lowest and highest *Min* satisfaction rates of the existing entity pairs are identical, indicating "perfect" fairness. Also, MMR 0 means no fairness in the bandwidth isolation guarantee.

Fig. 16 shows the MMR by changing the total *Min* (x-axis of Fig. 16a) and the number of tenants (x-axis of Fig. 16b). We show only the results of the hybrid type to highlight the fairness of the bandwidth guarantee where various bandwidth request types are mixed. We find that MMR for other bandwidth requirement types exhibits a similar tendency to Fig. 16, omitted here for brevity.

In Fig. 16a, TeaVisor always shows an MMR of near 1 over the total *Min*. For Libera+RL, up to 62.5% of the total *Min*, MMR is maintained as 0.98. However, as the total *Min* increases from 62.5% to 100%, the MMR decreases to 0.21, which is 4.7× poorer than TeaVisor. TeaVisor guarantees bandwidth (near MMR 1) for every entity pair through *Min* and *Max* assurance (Section 3.3) by multipath routing. For Libera and Libera+RL, due to the lack of bandwidth isolation, some entity pairs can monopolize the physical network bandwidth, which severely hinders fairness. Additionally, the MMR of Libera continuously decreases as the total *Min* increases. Specifically, when the total *Min* approaches 100%, MMR decreases to 0.05, which is 19.6× poorer than TeaVisor. This is because, as the total *Min* increases, the total bandwidth requirement of competing entity pairs increases. Thus, the fairness of Libera+RL and Libera decreases as the competing entity pairs increase.

In Fig. 16b, by varying the number of tenants, TeaVisor shows the highest MMR among the three NHs. Specifically, TeaVisor shows a near-constant MMR of 0.92. For Libera +RL, the MMR is almost the same as TeaVisor up to two tenants. However, when the number of tenants is more than two, its MMR decreases continuously to 0.21 (4.29× lower than the MMR of one tenant in Libera+RL). When comparing TeaVisor and Libera+RL, the MMR of Libera+RL is up to 4.65× lower than that of TeaVisor. The results demonstrate that TeaVisor provides significantly better fairness than Libera+RL. Lastly, Libera shows the worst MMR among the three NHs. Even for one tenant, the MMR of Libera is 0.11, which is 8.41× poorer than TeaVisor. The average MMR of Libera is 0.08, which is 12.1× poorer than TeaVisor. The reasons for the results are similar to the explanation in the above paragraph.

4.3 Effects of *Max* Assurance Policies

Fig. 17 shows the amount of additionally reserved bandwidth for *Max* assurance policies (i.e., ES, PBR, and PNU explained in Section 3.3.2). We investigate the effect of the policies. The experiment settings are explained in Section 4.1. The x-axis represents each entity pair, and the y-axis shows the additional throughput reserved by policies. The graphs in Fig. 17 are for each entity pair when the policies are applied. Because the experiment setting is made of eight entity pairs (i.e., 1, 4, 5, 8, 9, 12, 13, and 16) consuming 90% of the total *Min* (consuming high bandwidth), and of the other eight, the remaining 10% (consuming low bandwidth), the graphs in Fig. 17 look like step functions. In addition, the line without marks is the amount of *Min*.

Fig. 17 shows that PBR and PNU have similar patterns because TeaVisor guarantees the *Min* of entity pairs through *Min* assurance (Section 3.3.1) so that the actual usages of all entity pairs reach their *Min*. However, ES is different in that the additional throughput is less than those for PBR and PNU in entity pairs consuming high bandwidth and more with

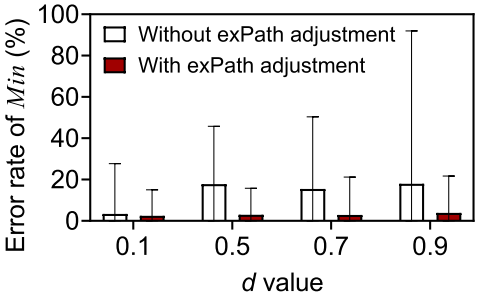


Fig. 18. Error rates of *Min* when changing *d* with or without exPath adjustment.

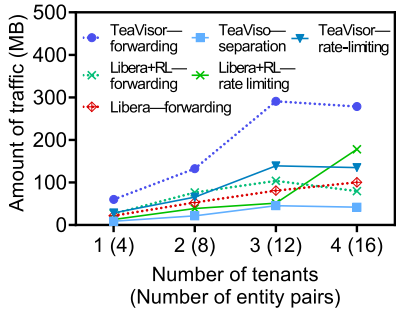


Fig. 19. Comparison of control traffic.

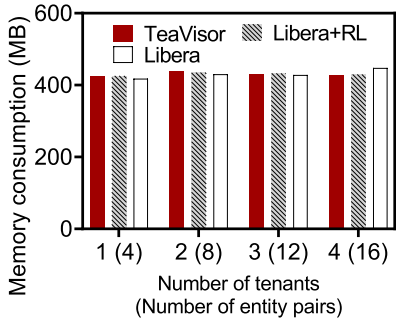


Fig. 20. Comparison of memory consumption.

entity pairs consuming low bandwidth. On average, however, the sum of the additional throughputs gained by the entity pair between the three policies differs by only 0.2%. This means that the policies use all the available capacity of the physical network and do not have much difference.

Fig. 17 shows that PBR and PNU have similar patterns. However, ES is different in that the additional throughput is less than those for PBR and PNU in entity pairs consuming high bandwidth and more with entity pairs consuming low bandwidth. On average, however, the sum of the additional throughputs gained by the entity pair between the three policies differs by only 0.2%, which means that the policies use all of the available capacity of the physical network and do not have much difference.

4.4 Effects of Parameter *d* and exPath Adjustment

Fig. 18 shows the error rates of *Min* by changing *d* to 0.1, 0.5, 0.7, and 0.9 (x-axis) when the exPath adjustment is on and off. The bars represent the average error rates of *Min*, and the whiskers indicate the range. We show the evaluation results for the [*Min*, *Inf*] type, but the other bandwidth requirement types show similar tendencies.

We first explain the results without the exPath adjustment (white bars). When *d* is 0.1, the average error rate is less than 4% (3.34%). When *d* ranges from 0.5 to 0.9, the average error rates jump to 17.96%. As *d* increases, the bandwidth to be secured for mPath increases. So, it is difficult to secure sufficient bandwidth, which increases the error rates. Moreover, with exPath adjustment (red-colored bars), the error rate is much lower than without exPath adjustment. Specifically, exPath keeps the average error rates below 4%, which improves the average error rates by 5.32×. This is because exPath adjustment changes the bandwidth given to the exPaths so that more mPaths can be secured.

Furthermore, regarding the peak error rates (whiskers in the graph), the peak errors without exPath adjustment increase up to 91.93% when *d* is 0.9. This is a 3.32× increase over the error rate when *d* is 0.1. On the other hand, with the exPath adjustment, the peak error rate does not increase much. In summary, exPath adjustment keeps low the average error rates, regardless of *d*.

4.5 Overheads

4.5.1 Control Traffic Consumption

Fig. 19 shows the total amount of control traffic generated for installing the forwarding, separation, and rate-limiting rules. In addition to the forwarding rule, Libera+RL installs a rate-limiting rule, and TeaVisor installs separation rules and rate-limiting rule to satisfy the bandwidth isolation guarantee. Moreover, TeaVisor installs multipaths (e.g., two to three paths in our case), but Libera and Libera+RL install only one path. Therefore, TeaVisor inevitably consumes more control traffic for the flow rule installation. As a result, Fig. 19 shows that TeaVisor consumes 2.6× and 2.9× more control traffic than Libera +RL and Libera, respectively. This overhead is unavoidable for TeaVisor to provide substantial improvements on bandwidth isolation guarantee. However, considering that the flow rule installations occur at once for a path between entity pairs, we believe that this overhead does not affect the throughput.

4.5.2 Additional Memory Consumption

Fig. 20 shows the average memory consumption. It is expected that TeaVisor consumes additional memory for path chart (e.g., vPath and pPath objects). Therefore, we optimize the internal object management of TeaVisor (e.g., avoiding redundant object allocation). As a result, TeaVisor consumes 99.7% of Libera's memory consumption on average; so, the memory consumption of TeaVisor is similar to those of existing methods that do not provide the bandwidth isolation guarantee.

5 Discussion

We discuss further research issues related to TeaVisor:

Scalability. As SDN-NV manages networks via NH, it could become a bottleneck in terms of scalability. This is similar to TeaVisor in that it provides bandwidth isolation guarantee in NH. In fact, this problem is a fundamental challenge in SDN structures that puts network control via a central entity. As a solution, several studies and SDN

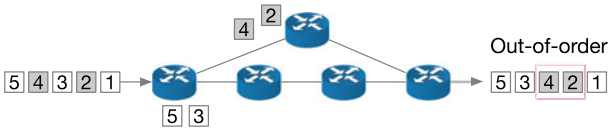


Fig. 21. Example of out-of-order packet transmission for multipath routing.

controllers suggest a physically distributed but logically central structure. This architecture has also been designed for NH [47]. Thus, we expect TeaVisor to be readily extended to such architectures.

Applicability to SDN. One might be concerned about whether TeaVisor can work with non-virtualized SDN. We think it is possible because there is already an approach [7] to use SDN-NV for managing a physical network with existing SDN controllers. The components of TeaVisor can act as a proxy between the SDN controller and the physical network.

Security of Switches. As illustrated in [48], suppose there are less trusted switches. Then, a tenant might want to avoid the switches in its mPaths. For these situations, TeaVisor can be extended to selectively turn off its operations, such as path virtualization and bandwidth reservation. As we have already designed and implemented the selective operation of NH on flow rules in [40], we believe such an extension is possible.

Out-of-Order Packets. Multipath routing can result in unexpected throughput due to out-of-order packets. The example in Fig. 21 demonstrates a single flow transmitted by two different paths. When the two paths have different hop counts, the order of packets arriving at the destination host can be twisted [49]. When out-of-order packets are detected, the host significantly reduces the window size (e.g., decreased by half in TCP), which results in the throughput downgrade. So how the out-of-order problem affects TeaVisor is an open question.

For the simple evaluation, we set up a topology in which the hop counts of pPaths are not equal. We assume that TeaVisor selects multiple paths of five hops. For comparison, shortest-path routing method (e.g., routing information protocol) is used because it is the most popular. Note that the shortest-path routing prioritizes the paths with the shortest hop counts. We send 1000 TCP packets and measure each packet's arrival time (packet receive time). Out of 1000 packets, TeaVisor shows only 33 out-of-order packets, whereas shortest-path routing shows 218, which is 6.6× higher. Consequently, TeaVisor accomplishes flow completion within 21.3 s, whereas shortest path routing takes 26.9 s, which is 1.26× longer. The results mean that TeaVisor does not solve the out-of-order problem completely, but can help relieve the problem. We would like to point out that solving the out-of-order packet problem is a separate research topic. In fact, there are various previous studies (e.g., [50], [51], [52]) that have tried to solve the out-of-order problem. So, we leave solving the out-of-order problem for future work.

6 Conclusion

This paper presents TeaVisor, the first bandwidth isolation guarantee framework for SDN-NV with three components: path virtualization, bandwidth reservation, and path establishment. We evaluate TeaVisor for the error rate of bandwidth isolation guarantee and overheads with comprehensive experiments. We find that TeaVisor achieves near-zero error rates for both the minimum and maximum bandwidth requirements. In addition, through implementation optimization, we show that TeaVisor consumes no additional memory compared to existing NHs.

TeaVisor deals with the bandwidth isolation for the data plane. In reality, control traffic [53], [54] that consists of the control plane can consume bandwidth, which can affect the accuracy of the bandwidth isolation of TeaVisor. We plan to work on predicting the control traffic so that TeaVisor can factor in the bandwidth of the control plane. We believe that such a prediction can render the bandwidth isolation of TeaVisor practically meaningful.

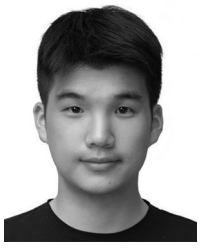
Acknowledgments

We thank Heesang Jin and Minkoo Kang, listed in [1], who made substantial contributions to earlier work. Gyeongsik Yang and Chuck Yoo are co-corresponding authors. A preliminary version of this article appeared in the proceedings of IEEE INFOCOM 2021 - IEEE Conference on Computer Communications [1].

References

[1] G. Yang, Y. Yoo, M. Kang, H. Jin, and C. Yoo, "Bandwidth isolation guarantee for SDN virtual networks," in *Proc. IEEE Conf. Comput. Commun.*, 2021, pp. 1–10.
[2] A. Rodriguez-Natal et al., "Global state, local decisions: Decentralized NFV for ISPs via enhanced SDN," *IEEE Commun. Mag.*, vol. 55, no. 4, pp. 87–93, Apr. 2017.
[3] C. Janz, L. Ong, K. Sethuraman, and V. Shukla, "Emerging transport SDN architecture and use cases," *IEEE Commun. Mag.*, vol. 54, no. 10, pp. 116–121, Oct. 2016.
[4] R. Sherwood, G. Gibb, K.-K. Yap, G. Appenzeller, N. Mckeown, and G. Parulkar, "Can the production network be the testbed?," in *Proc. 9th USENIX Symp. Operating Syst. Des. Implementation*, 2010, pp. 365–378.
[5] D. Drutskoy, E. Keller, and J. Rexford, "Scalable network virtualization in software-defined networks," *IEEE Internet Comput.*, vol. 17, no. 2, pp. 20–27, Mar./Apr. 2013.
[6] A. Al-Shabibi et al., "OpenVirteX: Make your virtual SDNs programmable," in *Proc. 3rd Workshop Hot Topics Softw. Defined Netw.*, 2014, pp. 25–30.
[7] X. Jin, J. Gossels, J. Rexford, and D. Walker, "CoVisor: A compositional hypervisor for software-defined networks," in *Proc. 12th USENIX Symp. Netw. Syst. Des. Implementation*, 2015, pp. 87–101.
[8] G. Yang, B.-Y. Yu, S.-M. Kim, and C. Yoo, "LiteVisor: A network hypervisor to support flow aggregation and seamless network reconfiguration for VM migration in virtualized software-defined networks," *IEEE Access*, vol. 6, pp. 65 945–65 959, 2018.
[9] G. Yang, B.-Y. Yu, H. Jin, and C. Yoo, "Libera for programmable network virtualization," *IEEE Commun. Mag.*, vol. 58, no. 4, pp. 38–44, Apr. 2020.
[10] Y. Yoo, G. Yang, M. Kang, and C. Yoo, "Adaptive control channel traffic shaping for virtualized SDN in clouds," in *Proc. IEEE 13th Int. Conf. Cloud Comput.*, 2020, pp. 22–24.
[11] G. Yang, C. Shin, Y. Yoo, and C. Yoo, "A case for SDN-based network virtualization," in *Proc. 29th Int. Symp. Model. Anal. Simul. Comput. Telecommun. Syst.*, 2021, pp. 1–8.
[12] P. Costa, M. Migliavacca, P. Pietzuch, and A. L. Wolf, "NaaS: Network-as-a-service in the cloud," in *Proc. 2nd USENIX Workshop Hot Topics Manage. Internet Cloud Enterprise Netw. Serv.*, 2012.
[13] M. Kang, G. Yang, Y. Yoo, and C. Yoo, "TensorExpress: In-network communication scheduling for distributed deep learning," in *Proc. IEEE 13th Int. Conf. Cloud Comput.*, 2020, pp. 25–27.
[14] G. Yang, C. Shin, J. Lee, Y. Yoo, and C. Yoo, "Prediction of the resource consumption of distributed deep learning systems," *Proc. ACM Meas. Anal. Comput. Syst.*, vol. 6, no. 2, pp. 1–25 Jun. 2022.
[15] M. Chowdhury, Z. Liu, A. Ghodsi, and I. Stoica, "HUG: Multi-resource fairness for correlated and elastic demands," in *Proc. 13th USENIX Symp. Netw. Syst. Des. Implementation*, 2016, pp. 407–424.

[16] A. Shieh, S. Kandula, A. Greenberg, C. Kim, and B. Saha, "Sharing the data center network," in *Proc. 8th USENIX Symp. Netw. Syst. Des. Implementation*, 2011, pp. 309–322.
[17] K. Jang, J. Sherry, H. Ballani, and T. Moncaster, "Silo: Predictable message latency in the cloud," in *Proc. ACM Conf. Special Int. Group Data Commun.*, 2015, pp. 435–448.
[18] J. C. Mogul and L. Popa, "What we talk about when we talk about cloud network performance," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 42, no. 5, pp. 44–48, 2012.
[19] L. Popa, G. Kumar, M. Chowdhury, A. Krishnamurthy, S. Ratnasamy, and I. Stoica, "FairCloud: Sharing the network in cloud computing," in *Proc. ACM SIGCOMM Conf. Appl. Technol. Archit. Protoc. Comput. Commun.*, 2012, pp. 187–198.
[20] 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. ACM SIGCOMM Conf. SIGCOMM*, 2013, pp. 351–362.
[21] S. Hu, W. Bai, K. Chen, C. Tian, Y. Zhang, and H. Wu, "Providing bandwidth guarantees, work conservation and low latency simultaneously in the cloud," *IEEE Trans. Cloud Comput.*, vol. 9, no. 2, pp. 763–776, Apr-Jun 2021.
[22] F. Liu, J. Guo, X. Huang, and J. C. Lui, "eBA: Efficient bandwidth guarantee under traffic variability in datacenters," *IEEE/ACM Trans. Netw.*, vol. 25, no. 1, pp. 506–519, Feb. 2017.
[23] T. Lam, S. Radhakrishnan, A. Vahdat, and G. Varghese, "NetShare: Virtualizing data center networks across services," Dept. Comput. Sci. Eng., Univ. California, Los Angeles, USA, Tehc. Rep. CS2010-0957, 2010.
[24] K. Lee, C.-H. Hong, J. Hwang, and C. Yoo, "Dynamic network scheduling for virtual routers," *IEEE Syst. J.*, vol. 14, no. 3, pp. 3618–3629, Sep. 2020.
[25] D. S. Marcon, F. M. Mazzola, and M. P. Barcellos, "Achieving minimum bandwidth guarantees and work-conservation in large-scale, SDN-based datacenter networks," *Comput. Netw.*, vol. 127, pp. 109–125, 2017.
[26] A. Lee, P. Wang, S.-C. Lin, I. F. Akyildiz, and M. Luo, "Dynamic bandwidth allocation in SDN based next generation virtual networks: A deterministic network calculus approach," in *Proc. Conf. Res. Adaptive Convergent Syst.*, 2018, pp. 80–87.
[27] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat, "Hedera: Dynamic flow scheduling for data center networks," in *Proc. 7th USENIX Symp. Netw. Syst. Des. Implementation*, 2010, Art. no. 19.
[28] M. Alizadeh et al., "CONGA: Distributed congestion-aware load balancing for datacenters," in *Proc. ACM Conf. SIGCOMM*, 2014, pp. 503–514.
[29] N. Katta et al., "Clove: Congestion-aware load balancing at the virtual edge," in *Proc. 13th Int. Conf. Emerg. Netw. Experiments Technol.*, 2017, pp. 323–335.
[30] S. Ghorbani, Z. Yang, P. B. Godfrey, Y. Ganjali, and A. Firoozshahian, "DRILL: Micro load balancing for low-latency data center networks," in *Proc. Conf. ACM Special Int. Group Data Commun.*, 2017, pp. 225–238.
[31] J. Guo, F. Liu, T. Wang, and J. C. Lui, "Pricing intra-datacenter networks with over-committed bandwidth guarantee," in *Proc. USENIX Annu. Tech. Conf.*, 2017, pp. 69–81.
[32] G. Yang, H. Jin, M. Kang, G. J. Moon, and C. Yoo, "Network monitoring for SDN virtual networks," in *Proc. IEEE Conf. Comput. Commun.*, 2020, pp. 1261–1270.
[33] A. Khan, A. Zugenmaier, D. Jurca, and W. Kellerer, "Network virtualization: A hypervisor for the Internet?," *IEEE Commun. Mag.*, vol. 50, no. 1, pp. 136–143, Jan. 2012.
[34] G. Yang, Y. Yoo, M. Kang, H. Jin, and C. Yoo, "Accurate and efficient monitoring for virtualized SDN in clouds," *IEEE Trans. Cloud Comput.*, to be published, doi: 10.1109/TCC.2021.3089225.
[35] P. Kumar et al., "PicNIC: Predictable virtualized NIC," in *Proc. ACM Special Int. Group Data Commun.*, 2019, pp. 351–366. [Online]. Available: https://doi.org/10.1145/3341302.3342093
[36] J. Mudigonda, P. Yalagandula, J. Mogul, B. Stiekes, and Y. Pouffary, "NetLord: A scalable multi-tenant network architecture for virtualized datacenters," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 41, no. 4, pp. 62–73, 2011.
[37] W. Bai, L. Chen, K. Chen, D. Han, C. Tian, and W. Sun, "PIAS: Practical information-agnostic flow scheduling for data center networks," in *Proc. 13th ACM Workshop Hot Topics Netw.*, 2014, pp. 1–7.
[38] S. Hu et al., "Explicit path control in commodity data centers: Design and applications," in *Proc. 12th USENIX Symp. Netw. Syst. Des. Implementation*, 2015, pp. 15–28.
[39] Z. Zhang et al., "Hashing linearity enables relative path control in data centers," in *Proc. USENIX Annu. Tech. Conf.*, 2021, pp. 855–862.
[40] G. Yang, B.-Y. Yu, W. Jeong, and C. Yoo, "FlowVirt: Flow rule virtualization for dynamic scalability of programmable network virtualization," in *Proc. IEEE 11th Int. Conf. Cloud Comput.*, 2018, pp. 350–358.
[41] A. Fischer, J. F. Botero, M. T. Beck, H. De Meer, and X. Hesselbach, "Virtual network embedding: A survey," *IEEE Commun. Surveys Tuts.*, vol. 15, no. 4, pp. 1888–1906, Fourth Quarter 2013.
[42] B. Pfaf et al., "The design and implementation of open vSwitch," in *Proc. 12th USENIX Symp. Netw. Syst. Des. Implementation*, 2015, pp. 117–130.
[43] P. Berde et al., "ONOS: Towards an open, distributed SDN OS," in *Proc. 3rd Workshop Hot Topics Softw. Defined Netw.*, 2014, pp. 1–6.
[44] V. Gueant, "iPerf-the TCP, UDP and SCTP network bandwidth measurement tool," 2020. Accessed: Oct. 2022. [Online]. Available: https://iperf.fr/
[45] D. Shue, M. J. Freedman, and A. Shaikh, "Performance isolation and fairness for multi-tenant cloud storage," in *Proc. 10th USENIX Symp. Operating Syst. Des. Implementation*, 2012, pp. 349–362.
[46] H. Park, S. Yoo, C.-H. Hong, and C. Yoo, "Storage SLA guarantee with novel SSD I/O scheduler in virtualized data centers," *IEEE Trans. Parallel Distrib. Syst.*, vol. 27, no. 8, pp. 2422–2434, Aug. 2016.
[47] Y. Han et al., "ONVisor: Towards a scalable and flexible SDN-based network virtualization platform on ONOS," *Int. J. Netw. Manage.*, vol. 28, no. 2, 2018, Art. no. e2012.
[48] P. Narula, S. K. Dhurandher, S. Misra, and I. Woungang, "Security in mobile ad-hoc networks using soft encryption and trust-based multi-path routing," *Comput. Commun.*, vol. 31, no. 4, pp. 760–769, 2008.
[49] S. Huang, D. Dong, Z. Zhou, and X. Liao, "MP-CREDIT: Multi-path credit for high-speed data center transports," *Comput. Netw.*, vol. 193, 2021, Art. no. 108061.
[50] E. Vanini, R. Pan, M. Alizadeh, P. Taheri, and T. Edsall, "Let it flow: Resilient asymmetric load balancing with Flowlet switching," in *Proc. 14th USENIX Symp. Netw. Syst. Des. Implementation*, 2017, pp. 407–420.
[51] Y. Geng, V. Jeyakumar, A. Kabbani, and M. Alizadeh, "JUGGLER: A practical reordering resilient network stack for datacenters," in *Proc. 11th Eur. Conf. Comput. Syst.*, 2016, pp. 1–16.
[52] S. Kandula, D. Katabi, S. Sinha, and A. Berger, "Dynamic load balancing without packet reordering," *SIGCOMM Comput. Commun. Rev.*, vol. 37, no. 2, pp. 51–62, Mar. 2007.
[53] A. Bianco, P. Giaccone, R. Mashayekhi, M. Ullio, and V. Vercellone, "Scalability of ONOS reactive forwarding applications in ISP networks," *Comput. Commun.*, vol. 102, pp. 130–138, 2017.
[54] B.-Y. Yu, G. Yang, and C. Yoo, "Comprehensive prediction models of control traffic for SDN controllers," in *Proc. IEEE 4th Conf. Netw. Softwarization Workshops*, 2018, pp. 262–266.

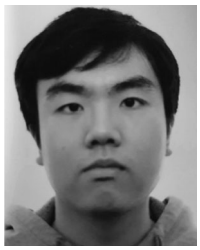


Yeonho Yoo (Graduate Student Member, IEEE) received the BS degree in computer science from Kookmin University, Seoul, South Korea, in 2017, and the MS degree in computer science from Korea University, Seoul, in 2021. He is currently working toward the PhD degree with Korea University. He is currently working as an intern with Microsoft Research Asia. His current research interests include network virtualization, SDN, datacenter systems, and AI systems.

Gyeongsik Yang (Member, IEEE) received the BS, MS, and PhD degrees in computer science from Korea University, Seoul, South Korea, in 2015, 2017, and 2019, respectively. He worked as a research intern with Microsoft Research Asia in 2018. He is currently a research professor with Korea University. His research interests include operating systems, network virtualization, AI systems, datacenter systems, and SDN.

Jeunghwan Lee received the BS degree in computer science from Korea University, Seoul, South Korea, in 2021. He is currently working toward the MS degree with Korea University. His current research interests include distributed deep learning systems, cloud orchestration, and SDN.

Changyong Shin received the BS degree in computer science from Korea University, Seoul, South Korea, in 2021. He is currently working toward the PhD degree with Korea University. His current research interests include distributed deep learning systems, cloud orchestration, and SDN.



Hoseok Kim is currently working toward the BS degree in computer science and engineering with Korea University, Seoul, South Korea. He is currently working as a research intern with the Operating Systems Lab, Korea University. His current research interests include network virtualization, datacenter systems, and SDN.

Chuck Yoo (Member, IEEE) received the BS and MS degrees in electronic engineering from Seoul National University, and the MS and PhD degrees in computer science from the University of Michigan, Ann Arbor. He worked as a researcher with Sun Microsystems. Since 1995, he has been with the College of Informatics, Korea University, where he is currently a professor. His research interests include server/network virtualization and operating systems.