Bandwidth Isolation Guarantee for SDN Virtual Networks

Gyeongsik Yang*, Yeonho Yoo*, Minkoo Kang*, Heesang Jin†, and Chuck Yoo*
*Department of Computer Science and Engineering, Korea University, Seoul, Republic of Korea
†Blockchain Research Section, Electronics and Telecommunications Research Institute (ETRI), Daejeon, Republic of Korea
ksyang@os.korea.ac.kr, yhyoo@os.korea.ac.kr, mkkang@os.korea.ac.kr, jinhs@etri.re.kr, chuckyoo@os.korea.ac.kr

***Abstract*—We introduce TeaVisor, which provides bandwidth isolation guarantee for software-defined networking (SDN)-based network virtualization (NV). SDN-NV provides topology and address virtualization while allowing flexible resource provisioning, control, and monitoring of virtual networks. However, to the best of our knowledge, the bandwidth isolation guarantee, which is essential for providing stable and reliable throughput on network services, is missing in SDN-NV. Without bandwidth isolation guarantee, tenants suffer degraded service quality and significant revenue loss. In fact, we find that the existing studies on bandwidth isolation guarantees are insufficient for SDN-NV. With SDN-NV, routing is performed by tenants, and existing studies have not addressed the overloaded link problem. To solve this problem, TeaVisor designs three components: path virtualization, bandwidth reservation, and path establishment, which utilize multipath routing. With these, TeaVisor achieves the bandwidth isolation guarantee while preserving the routing of the tenants. In addition, TeaVisor guarantees the minimum and maximum amounts of bandwidth simultaneously. We fully implement TeaVisor, and the comprehensive evaluation results show that near-zero error rates on achieving the bandwidth isolation guarantee. We also present an overhead analysis of control traffic and memory consumption.**

***Index Terms*—SDN, Software-defined Networking, Network Virtualization, Bandwidth, Isolation, Performance**

I. Introduction

Software-defined networking (SDN) has made network systems open and softwarized and has evolved in many directions, such as network function virtualization [1] and transport SDN [2]. Among them, combinations of network virtualization (NV) and SDN [3]–[10] (SDN-NV) have been promoted. 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[2]. With such benefits, a tenant can build a custom VN topology of network switches and monitor their performance bottlenecks [11]. Additionally, within its VN, the tenant can process network connections with priorities for the purpose of quality-of-experience enhancements [8].

Datacenter applications exist on a broad scale, from commodity applications (e.g., video streaming and distributed file systems) to machine learning services (e.g., training and inference of neural networks) [12]–[14]. Many of these applications require network performance isolation. That is, tenants on a shared network should be able to use the expected network bandwidth[3] [15]. Performance isolation is typically expressed in terms of bandwidth requirements. When these requirements are not satisfied, service quality can be severely degraded [16], and a significant loss in revenue occurs [17], [18]. Therefore, the lack of the bandwidth isolation guarantee is a critical problem, yet SDN-NV poses significant challenges to support bandwidth isolation guarantee.

SDN-NV allows SDN controllers of tenants to calculate paths for entity pairs, making the problem difficult for existing studies on bandwidth isolation guarantee to be applied. Most existing studies [17]–[20] do not consider routing, which determines the path between an entity pair. Instead, they use the hose model, in which virtual machines are connected via a non-blocking virtual switch. This approach assumes that a path is given (solved by an orthogonal method) and focuses on the entity pair bandwidth. In SDN-NV, when multiple controllers calculate various paths, each path has a bandwidth requirement so that some links can become overloaded beyond the link capacity. However, existing studies do not address such issues.

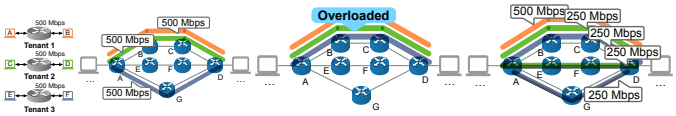
For example, Fig. 1a shows the bandwidth isolation guarantee in existing studies based on the hose model. First, suppose that three tenants (tenants 1, 2, and 3) have one entity pair each and express their bandwidth requirements—500 Mbps of traffic for each entity pair through the hose model (Fig. 1a, left), where all physical links can transmit 1 Gbps traffic. Then, according to the bandwidth requirements, suppose that the paths of tenants 1, 2, and 3 resulting from the orthogonal

This work was done while Heesang Jin was at Korea University. This work was partly supported by Institute of Information & Communications Technology Planning & Evaluation grant funded by the Korea government (No. 2015-0-00280, (SW Starlab) Next generation cloud infra-software toward the guarantee of performance and security SLA). This work was also supported by National Research Foundation of Korea funded by the Ministry of Science, ICT (No. NRF-2019H1D8A2105513). Corresponding author: Chuck Yoo.

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

[2]Without any explicit designation, we use the term "entity" to indicate the user's computing entities, such as virtual machines and containers. So, entity pair means the pair consisting of the 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) Existing studies with hose model.

(b) SDN-NV with hose model based studies.

(c) Solution of TeaVisor.

Fig. 1: Bandwidth isolation guarantee examples.

routing methods are A-B-C-D, A-B-C-D, and A-G-D, respectively. Then, existing studies make each entity pair consume bandwidth per their bandwidth requirements. Thus, for tenants 1 and 2, which share one path, each is limited to using that path at 500 Mbps, whereas tenant 3 uses another 500 Mbps, resulting in the satisfaction of all tenants' bandwidth requirements.

However, in SDN-NV, bandwidth isolation can be broken due to the routing of tenants. In Fig. 1b, suppose that the three tenants have the same bandwidth requirements and physical link capacities as Fig. 1a. Because the controllers do not know each other in the SDN-NV scheme, SDN controllers of tenants 1, 2, and 3 can possibly calculate the same forwarding paths consisting of switches A-B-C-D in Fig. 1b. In this situation, the physical links in the path become overloaded because the total bandwidth requirements (1.5 Gbps) exceed the link capacity of 1 Gbps. Therefore, the requested bandwidth requirements are not fully satisfied, even if the physical network has other paths (i.e., A-E-F-D and A-G-D), which remain idle.

Some existing studies are not based on the hose model, but they provide fair sharing between source and destination entities [17], [21]–[23]. For example, Seawall [21] pairs each source-end entity with a weight so that the share of bandwidth for the end entity is proportional to the paired weight of all network links. Here, 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. In addition, the bandwidth given to an end entity can be arbitrarily reduced [18] by the weights of other network traffic, so the bandwidth isolation guarantee cannot be achieved in SDN-NV.

On the other hand, bandwidth allocation methods on non-virtualized SDN, such as Predictor [24] and DBS [25], are not directly related to the bandwidth isolation guarantee when using SDN-NV. Predictor is used to minimize the flow table size and provide the predictable performance of switches and thereby flows. DBS focuses on scheduling within switches to dynamically allocate data rates to the flows. The problem that these methods proposed to solve is rate limiting within switches. Therefore, their algorithms do not address bandwidth isolation for SDN-NV.

Thus, we propose TeaVisor to provide the bandwidth isolation guarantee for SDN-NV. The key idea of TeaVisor is to associate flow rules from the SDN controller with bandwidth requirements and enable multipath routing to achieve the bandwidth requirements. For example, as a solution to the problem shown in Fig. 1b, TeaVisor establishes additional paths for the bandwidth isolation guarantee (Fig. 1c) as follows. For the same path, A-B-C-D, shared by all tenants, tenant 1 is allocated its required 500 Mbps by occupying half of the link capacity, whereas tenants 2 and 3 achieve half of the bandwidth requirement, 250 Mbps. For the remaining bandwidth requirements (i.e., 250 Mbps for tenants 2 and 3), TeaVisor introduces path virtualization to establish additional paths needed to satisfy 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 [26]–[29], but they are not applicable to SDN-NV. We discuss their limitations and the novelty of TeaVisor in §II-C. To the best of our knowledge, TeaVisor is the first SDN-NV study that considers multipath routing for bandwidth isolation guarantee.

To make it practically applicable, TeaVisor is designed to accept two types of bandwidth requirements: minimum and maximum, as these reflect the typical requirements of datacenters [15] for achieving both bandwidth isolation and enhanced resource unitization. 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 for work conserving. In a typical datacenter, tenants follow the pay-as-use policy, so it is important to adhere to the upper limit to which the bandwidth can be allocated. However, according to [30], most datacenters cannot guarantee an upper limit. As a result, tenants are overcharged. TeaVisor ensures that both the minimum and maximum bandwidth requirements are satisfied.

The bandwidth isolation guarantee of TeaVisor is achieved by three components: path virtualization, bandwidth reservation, and path establishment. First, path virtualization (§III-B) is for crafting the intended virtual path from flow rules and for creating the physical paths that realize the virtual path in the physical network. Then, the bandwidth reservation (§III-C) reserves the bandwidth requirement on physical paths[4] at the link level. Third, path establishment (§III-D) substantiates the physical paths using flow rules, such as flow rules for packet splitting on multiple paths and for rate limiting.

TeaVisor is implemented as a complete system based on Libera [8], an open-source network hypervisor (NH). We conduct comprehensive evaluations, including the satisfaction of bandwidth requirements, policies, and overhead, by varying the amount of bandwidth requirements and the number of tenants. In short, TeaVisor makes the following contributions:

- Being the first bandwidth isolation guarantee for SDN-NV.
- Introducing path virtualization and bandwidth reservation for multipath routing in SDN-NV.
- Achieving near-zero error rates for the bandwidth isolation guarantee and analyzing the overhead of the bandwidth isolation guarantee.

II. Background and Related Work

A. SDN-NV

SDN-NV virtualizes tenant networks via NH [3], [5], [8], [31], which abstracts the underlining (physical) network as

[4]A path consists of links between switches.

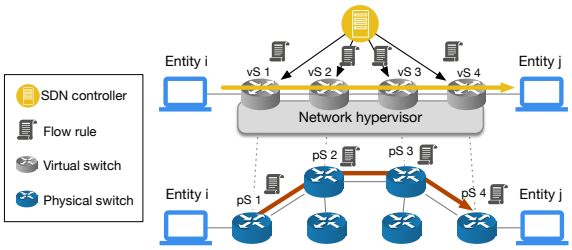


Fig. 2: Relationship between a path and flow rules.

VNs and provides them to SDN controllers. Compared with SDNs, there are two main characteristics of SDN-NV: topology and address virtualization. Topology virtualization allows the tenants to configure their arbitrary VN topology, which can quite differ from the physical network. By using the NH, the number of switches, ports, and link connections comprising a VN topology can be freely configured. When the VN is created with its virtual switches, 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 [8]. Also, through address virtualization, the NH provides a virtual address that allows each tenant to freely choose their addressing scheme without worrying about conflicts with other tenants.

Fig. 2 explains the structure of SDN-NV: the top portion represents the VN, and the bottom portion represents the physical network. Each VN consists of end entities (e.g., entities A and B), virtual switches (e.g., vS 1, 2, 3, and 4), and virtual links that connect the virtual switches. When new network traffic arrives, an event is delivered to the SDN controller passing through the NH. Then, the SDN controller calculates a path (routing) between entities for the traffic (e.g., vS1 to 4 in Fig. 2), and then sends flow rules (for individual virtual switches) to NH. Upon receiving the flow rules, the NH translates and installs each flow rule while considering the address of the entities and virtual topology of the mapped physical switches (e.g., pS 1 to 4) [32]. This example shows that all network controls and operations are based on the flow rule translation on NH.

B. Bandwidth Isolation Guarantee in SDN-NV

Because bandwidth is the critical performance metric of network services, existing SDN-NV studies attempted to address this issue. For example, FlowVisor [3] attempted to isolate the bandwidth of each tenant. This isolation is achieved by multiple priority queues at each switch, which is a common feature of existing switches. By mapping the traffic of end pairs to each queue, FlowVisor can only achieve bandwidth management per queue. Therefore, this scheme is not really a bandwidth isolation guarantee because the switch has a limited number of queues (e.g., usually two to eight) [33]. Typically, a single tenant creates tens-of-hundreds of entities [34]. Thus, the number of entity pairs easily exceeds the number of queues. To the best of our knowledge, no NHs have yet solved the bandwidth isolation guarantee problem.

C. Bandwidth Isolation Guarantee and Multipath Routing

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

1) Bandwidth isolation guarantee: Existing studies of bandwidth isolation can be categorized into two types: with hose model or without hose model. Studies with hose model [17]–[20] achieve the bandwidth isolation guarantee. However, Studies without hose model [17], [21]–[23] do not completely satisfy the guarantee—instead, the studies use proportional share [17], switch queueing [19], or rate limiting on the path [18], [20]–[23] with varying bandwidth isolation units (e.g., per-tenant, per-entity, per-entity pair, and per-virtual router management). In addition, there are various methods to achieve rate limiting, including priority queueing [19], logistical modeling [20], and credit-based modeling [23]. However, none of these existing studies are applicable in solving the overloaded link problem illustrated in Fig. 1b because SDN-NV allows tenants' SDN controllers to perform routing. Existing studies are summarized in Table I.

2) Multipath routing: Previous multipath routing [26]–[29] studies focus on improving network utilization by splitting traffic into multiple paths. These studies established traffic splitting through a central controller [26], at network hops [27], [29], and on software switches [28]. They also use global network information [26], congestion information [27], and local/partial information [28], [29]. Although they are successful in improving network utilization, these studies do not attain the bandwidth isolation guarantee. Moreover, they provide routing schemes that create paths solely. This means that SDN-NV allows SDN controllers of tenants to perform routing. Thus, SDN-NV cannot use the previous multipath routing scheme because the paths calculated from SDN controllers and multipath routing studies cannot work independently of each other. TeaVisor solves this problem through path virtualization.

III. Design

A. Workflow

Fig. 3 shows the architecture of TeaVisor. TeaVisor enables each tenant to specify bandwidth requirements for its end entities (❶-1). So, bandwidth requirements are entered per entity pair. TeaVisor supports two types of bandwidth requirements: minimum and maximum (*Min* and *Max*). Also, TeaVisor collects physical network information (e.g., total and remaining link capacity) via network monitoring (❶-2).

The workflow of TeaVisor is as follows. When a new packet flows between an entity pair, the physical network notifies TeaVisor of the new traffic as an event (❷-1), and the event is delivered to the 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 flow rules, path virtualization (§III-B) crafts a virtual path (vPath) (❹). The vPath is an end-to-end forwarding path for an entity pair consisting of a number of virtual switches and links.

Then, the bandwidth reservation component (§III-C) fulfills the bandwidth isolation guarantee (❺) by utilizing multiple

TABLE I: Bandwidth isolation guarantee–related work comparison.

Study	VN Model	Bandwidth isolation guarantee	Bandwidth isolation unit	Method	Overloaded link problem
FairCloud PS-L/N [17]	NA	No	Per-tenant or -entity	Proportional share	Not solved
FairCloud PS-P [17]	Hose model	Yes	Per-entity	Queueing in tree topology	
ElasticSwitch [18]	Hose model	Yes	Per-entity pair	Rate allocation, limiting	
Trinity [19]	Hose model	Yes	Per-entity pair	Switch priority queuing ECN	
eBA [20]	Hose model	Yes	Per-entity pair	Rate control algorithm	
Seawall [21]	NA	No	Per-source entity	Weight-based rate limiting	
NetShare [22]	NA	No	Per tenant	Weight-based rate limiting	
CreditBank [23]	Virtual router model	No	Per virtual router	Credit-based rate limiting	
TeaVisor	**SDN-NV**	**Yes**	**Per-entity pair**	**Multipath routing**	**Solved**

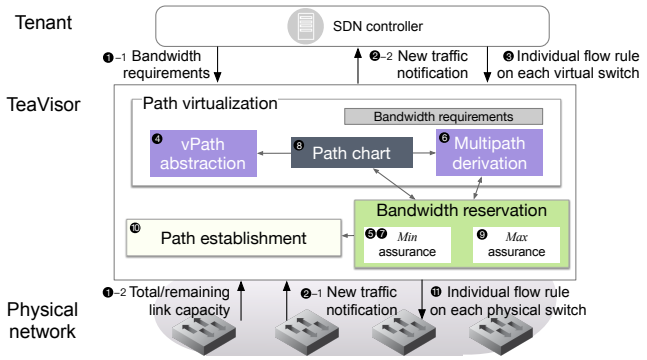


Fig. 3: TeaVisor architecture and workflow.

physical paths (pPaths) and reserving bandwidths' *Min* requirements on switches and links that comprise the pPath. This component works with the path virtualization component (❻-❼). Note that TeaVisor maintains the mappings of vPath and pPaths as path chart (§III-B3) for the tenant's network management (❽). After reserving *Min*, the bandwidth reservation component periodically checks the idle network capacity and reserves the additional bandwidth for work conserving up to *Max* (❾). All paths and the reserved bandwidth are realized in the physical network via the path establishment component (❿-⓫). A key aspect of TeaVisor is that it satisfies both *Min* and *Max* of the vPath through the utilization of multiple pPaths. We use the expression "satisfying *Min* on a vPath or an entity pair" to mean "reserving *Min* amount for pPaths transmitting packets for the entity pair of the vPath."

B. Path Virtualization

Path virtualization crafts the vPath for an entity pair from flow rules and identifies pPaths for the vPath. The challenge of crafting a vPath is that TeaVisor cannot obtain an end-to-end forwarding path directly from SDN controllers. What TeaVisor gets from the SDN controllers are flow rules to be installed on switches. However, each flow rule only contains information about matching the packet header and transmitting packets from the in-port to the out-port within a switch, which makes it difficult for TeaVisor to determine a forwarding path. As a solution, TeaVisor introduces a new scheme called "vPath abstraction" to craft a virtual path from flow rules.

1) vPath Abstraction: Fig. 4 illustrates the process of vPath abstraction. From flow rules from SDN controllers, TeaVisor connects the flow rules from start to end (i.e., source entity to destination entity) via rule chaining (explained below). Specifically, when a new flow rule arrives, TeaVisor groups the rule according to the path to which it belongs. Path virtualization generates a "pathID," an ID for each vPath, from the flow rule (① of Fig. 4). The pathIDs are used to identify both vPaths and entity pairs. The pathID is a combination of the tenant identifier and IP addresses of source and destination entities. TeaVisor checks whether a pathID is new, and if so, it indicates that a flow rule of a new vPath has arrived (②). Then, TeaVisor prepares a new vPath object to be created (②-1). Next, whenever a new flow rule arrives, it is checked whether the vPath of its pathID already exists (③).

If it does, the flow rule is used for rule chaining (④) as follows. Fig. 5 presents an example of rule chaining. Because each flow rule of a switch is intended for packet forwarding, it has the in-port and out-port information within a switch. For example, for switch 1 in Fig. 5, 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 way, TeaVisor traces flow rules one at a time, starting from the source to destination entities. When the end-to-end path is identified, the vPath abstraction is completed.

2) Multipath Derivation: Based on the vPath, multipath derivation identifies pPaths, a set of physical switches and links to deliver 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 "main path" (mPath). For example, in Fig. 2, mPath derived from the vPath consists of physical switches (e.g., pS 1 to 4) and the link that connects them. However, because mPath is determined based on vPath, the links of the mPath may become overloaded, as shown in Fig. 1b. To address this issue, TeaVisor uses multipath routing, in which additional pPaths are derived for the vPath to satisfy the bandwidth requirement. We call the additional paths for satisfying the bandwidth requirement "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. The bandwidth reservation component (§III-C) achieves the bandwidth isolation guarantee by utilizing exPaths derived by this routing algorithm.

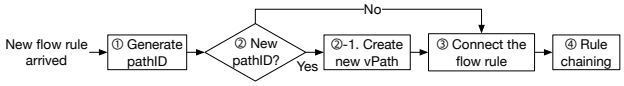


Fig. 4: vPath abstraction process.

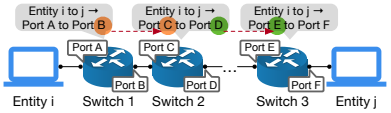


Fig. 5: Rule chaining example.

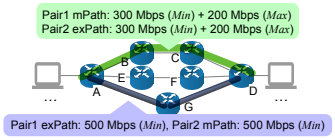


Fig. 6: Bandwidth reservation example.

3) Path Chart: The purpose of the path chart is to support the network management of SDN controllers (e.g., network monitoring) in the presence of the vPath and pPaths. The 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, the path chart determines the corresponding pPath flow rules. We use the mapping structures for the flow rules proposed in [35]. 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, the path chart provides the 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 [36] so that it delivers the aggregated values of multiple statistical values of pPaths as the statistics of a single vPath.

C. Bandwidth Reservation

The two bandwidth requirements, *Min* and *Max*, mean the following for vPath: *Min* is the amount of bandwidth that is always to be provided for the vPath, and up to *Max* bandwidth can be reserved to the vPath in a best-effort manner for work conserving. Tenants can request the following bandwidth requirement combinations: 1) *Min* with no *Max* (denoted [*Min, Min*]) and 2) *Min* with *Max* ([*Min, Max*])[5]. Note that the optimal solution to the bandwidth reservation problem in NV or SDN-NV is NP-hard [37] ; 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 changes, bandwidth reservation reruns for the changed values.

Fig. 6 shows an example of *Min* and *Max* assurance. We assume two entity pairs, pair1 whose mPath is A-B-C-D and pair2 with A-G-D as its mPath. Both have a bandwidth requirement of [800 Mbps, 1200 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 on exPath (A-G-D). For pair2, 500 Mbps is reserved on mPath (A-G-D) and the other 300 Mbps on exPath (A-B-C-D). After *Min* satisfaction, the algorithm for *Max* assurance runs periodically. It reserves additional bandwidth using the idle capacity of the physical network to both pPaths following the policies to be explained in §III-C2). In the example of Fig. 6, mPath and exPath each get 200 Mbps more. We explain *Min* assurance and *Max* assurance in detail next.

[5]Setting a value of *Max* as infinite, meaning that any additional bandwidth over the entity pair is welcomed.

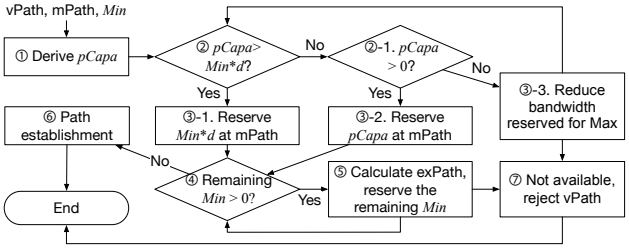


Fig. 7: *Min* assurance workflow.

1) Min Assurance: Min assurance achieves bandwidth isolation guarantee using multipath routing. The goal of *Min* assurance is to satisfy as many entity pairs as the physical network capacity permits by properly utilizing mPath and exPath. 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 vPath from tenants. However, there can be a case where the mPath of a new entity pair may not be able to transmit packets because the other entity pairs have already consumed all of the bandwidth of the links in the mPath. For example, in Fig. 7, assume that a new entity pair starts its traffic, its bandwidth requirement is [800 Mbps, 1200 Mbps], and mPath for the entity pair is A-B-C-D. In this situation, *Min* assurance cannot reserve bandwidth on mPath because the other entity pairs 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 *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. d is a tunable parameter of TeaVisor that determines the portion of *Min* to be reserved in mPath, ranging from zero to one.

Min assurance first derives *pCapa* of the given mPath (① of Fig. 7). *Min* assurance then checks that *pCapa* of mPath is sufficient for $Min*d$ (②). If *pCapa* satisfies $Min*d$ ($pCapa > Min*d$), $Min*d$ is reserved on mPath (③-1). Here, if d is 1, the entire amount of *Min* can be reserved to mPath. On the other hand, if d approaches 0, the bandwidth that can be reserved on 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), TeaVisor utilizes exPaths and reserves the *Min* on the exPaths (⑤). Specifically, exPath is created one at a time with the assistance of multipath derivation. If the *pCapa* of the created exPath ($pCapa^{exPath}$) is sufficient for *Min* ($pCapa^{exPath} > Min$), we reserve *Min* on exPath and complete exPath creation. However, if $pCapa^{exPath} < Min$, we reserve $pCapa^{exPath}$ on exPath, calculate remaining *Min* ($Min - pCapa^{exPath}$), and create another exPath to reserve the remaining *Min*. Routine ⑤ is repeated until the remaining *Min* becomes 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[6] (⑦). On the contrary, if $Min = 0$, which means that all of *Min* is reserved ("no" condition of ④), *Min* assurance is completed, and "path establishment" is called to install 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*. We then reserve the remaining *Min* on exPaths using the process starting from ④. However, if the *pCapa* is zero, it means that all bandwidth is reserved already. Therefore, we cannot reserve any bandwidth on mPath. We then first check whether the bandwidth of the link that decides *pCapa* has been reserved for the other entity pair's *Max* assurance (③-2). Assume that, in Fig. 6, a new mPath is A-B-C-D whose *pCapa*= 0. In this situation, the bandwidth reserved for *Max* can be reduced to secure bandwidth for a new mPath; because the bandwidth for *Max* is not mandatory, it can be dynamically regulated. On the other hand, if the bandwidth reserved for *Max* does not exist, this means that the physical network cannot accept the entity pair with *Min*, so we reject the entity pair's vPath (⑦). Then, in order to accept a greater number of entity pairs, a careful tuning of parameter d is required (e.g., smaller d for handling more entity pairs).

2) Max Assurance: *Max* assurance provides work conserving for TeaVisor. Note that *Max* assurance does not create or remove additional paths; instead, it only adds bandwidth to the existing pPaths. Algorithm 1 describes how *Max* assurance works. *Max* assurance runs at a regular interval to check whether there is any remaining bandwidth within the physical links (line 2 in Alg. 1). If a physical link (l_i) has idle bandwidth, Alg. 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:

- Equal share (ES, line 6): divide 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.

[6]Note that, like other bandwidth isolation guarantee or multipath routing studies, TeaVisor accepts and satisfies the bandwidth requirements to the extent that is acceptable to the physical network.

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

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

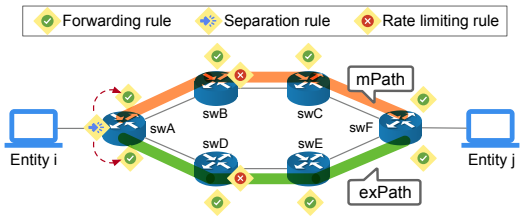


Fig. 8: Path establishment example.

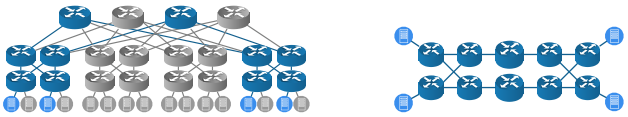
- Proportional share for actual network usage (PNU, line 8): divide the idle bandwidth based on the ratio of the monitoring results on the pPaths of P_n (actual throughput used within the reserved bandwidth).

For all three policies, *Max* assurance ensures that the total bandwidth reserved for an entity pair is lower than *Max* (line 5 and M functions on lines 7–9).

D. Path Establishment

Using path virtualization (§III-B) and bandwidth reservation (§III-C), TeaVisor derives multiple pPaths (i.e., an mPath and exPaths) and allocates the reserved bandwidth to them. The path establishment installs flow rules to substantiate bandwidth isolation in the physical network. Specifically, pPaths and their reserved bandwidths are converted into three types of flow rules: forwarding rule, separation rule, and rate limiting rule.

The forwarding rule is similar to the example found in Fig. 8 that delivers packets from the in-port to out-port within a switch. For mPath, flow rules are generated by translating flow rules sent by SDN controllers of tenants, which includes translation between virtual and physical networks (i.e., host address, switch address, in-port, and out-port). However, for exPaths, TeaVisor creates new flow rules because exPaths are created solely by TeaVisor. One of the new flow rules is the separation rule, which divides packets into multiple pPaths. Suppose that pPaths for an entity pair A and B are two pPaths:



(a) Physical network (gray) and mapped VN 1 (blue).

(b) VN topology layout.

Fig. 9: Network topology.

mPath of swA-swB-swC-swF and exPath of swA-swD-swE-swF, as shown in Fig. 8. Then, the separation rule is installed at swA where the two paths are split. Another new flow rule is the rate limiting rule. It limits packet transmission rate under the reserved bandwidth to prevent certain entity pairs from consuming excessive amounts of bandwidth. For the rate limiting rule, TeaVisor uses the meter table of OpenFlow (OF) 1.3. In Fig. 8, we install the rules for mPath and exPath on swB and swD, respectively.

IV. Implementation and Evaluation

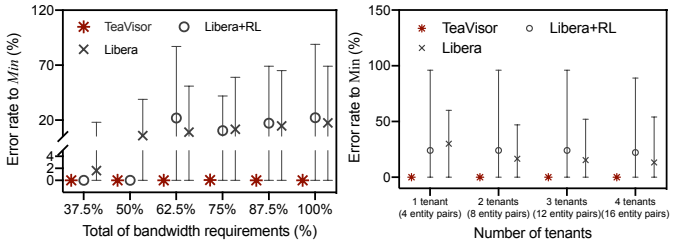
We implement all components of TeaVisor based on Libera (4.7K LoCs of Java). We use OF 1.3, the de-facto interface in SDN systems commonly supported by commercial SDN switches. TeaVisor implements separation and rate limiting rules using the group table and meter table mechanisms of OF 1.3. We empirically set the value of parameter d to 0.5 and perform network monitoring for *Max* assurance at every 5 s. We compare TeaVisor with 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 an existing algorithm of bandwidth isolation guarantee (i.e., rate limiting on each path as its *Min*) on SDN-NV. The purpose is to show that the existing studies are insufficient to handle the overloaded link problem.

A. Evaluation Environments

Three servers of Intel Xeon E5-2600 CPUs are deployed for our evaluations. They are connected through a 10 GbE switch. We run Mininet of Open vSwitches [38] for the physical network, and ONOS [39] is used as an SDN controller. As the physical network, we set a 4-ary fat-tree topology commonly used in datacenters (Fig. 9a). In addition, the VN topology is set as shown in Fig. 9b, consisting of 10 switches and four entities. In the evaluation, a tenant has four entity pairs, and each entity pair sends 128 TCP connections using iperf3 [40] at full speed. Bandwidth requirements are entered per entity pair. We conduct extensive measurements. However, owing to space limitations, only representative results are presented in this paper. However, the tendency of the omitted results is similar to the results shown. The evaluation of TeaVisor consists of three experiments.

1) Bandwidth Isolation Guarantee (§IV-B): We evaluate the bandwidth isolation guarantee by measuring the error rates on *Min* and *Max*, with changes to the amounts of bandwidth requirements and the number of tenants.



(a) Total amount of bandwidth requirements.

(b) Number of tenants.

Fig. 10: Error rates to *Min* ([*Min, Min*] case).

For these changes, we vary the total amount of *Min* (total *Min*) proportional to the entire network capacity (i.e., 37.5 to 100% because we find that the results under 37.5% are similar to that of 37.5%) with 16 entities of four tenants. For *Max*, we set *Max* as 120% of *Min* because it is more challenging to guarantee bandwidth isolation when the gap between *Max* and *Min* is small. We run experiments for two cases: [*Min, Min*] and [*Min, Max*]. For [*Min, Max*] (e.g., Fig. 13b), the x and the circle marks show *Min* and *Max*, respectively, given to each entity pair. Of the total *Min*, eight entity pairs (i.e., 1, 4, 5, 8, 9, 12, 13, and 16 entity pairs in Fig. 13b) are set to require 90% of the total *Min*, and the remaining eight entity pairs consume the other 10%.

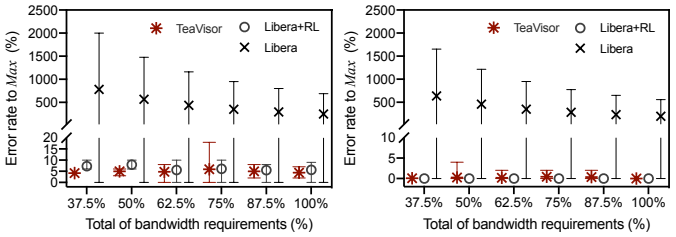
Regarding the number of tenants, we increase the number from one to four. Each tenant has four entity pairs. The total bandwidth requirement of one tenant is 25% of the physical network capacity. Thus, as the number of tenants increases to four, the number of entity pairs and the bandwidth requirement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

2) Effects of Max Assurance Policies (§IV-C): We analyze the effect of the three *Max* assurance policies (i.e., ES, PBR, and PNU) by measuring the additional bandwidth reserved for entity pairs. We present results of four tenants (16 entity pairs) having a total bandwidth requirement (*Min*) of 50% of physical network capacity. *Max* is set as infinite.

3) Overheads (§IV-D): We evaluate the overhead 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 all in 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.

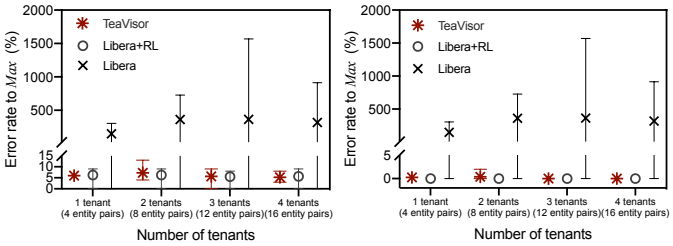
B. Bandwidth Isolation Guarantee

1) Minimum Bandwidth Requirements: In Fig. 10a and 10b, the y-axis is the error rate of *Min*, implying that the bandwidth is lower than *Min* (as error). The error rates are measured by varying the total amount of bandwidth requirements (Fig. 10a) and the number of tenants (Fig. 10b). An error rate of 0% means that the entity pair receives at least the *Min* amount of bandwidth during the experiments. The points and lines in Fig. 10 are the mean values and the range of error rates for TeaVisor, Libera+RL, and Libera, respectively. The results



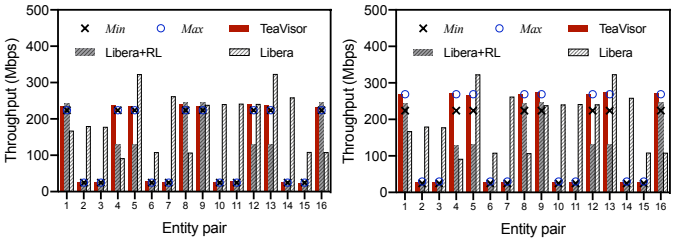
(a) [*Min, Min*] (b) [*Min, Max*]

Fig. 11: Error rates to *Max* by varying the number of bandwidth requirements.



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

Fig. 12: Error rates to *Max* by varying the number of tenants.



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

Fig. 13: Throughput according to *Min* and *Max*.

lacking lines indicate that the error rates for each entity pair are quite similar (e.g., TeaVisor's error rate at 37.5% is marked as a star in Fig. 10a, and its value is almost zero).

In Fig. 10a, TeaVisor, Libera+RL, and Libera show error rates on average of 0.01, 12, and 10%, respectively. We first compare TeaVisor with Libera+RL to illustrate the overloaded link problem. Up to the bandwidth requirement of 50%, TeaVisor and Libera+RL show similar error rates (0% on average). This is because physical links are not overloaded. However, once the bandwidth requirements exceed 50%, links become overloaded, so Libera+RL's error rate increases dramatically up to 89%.

Second, by comparing Libera+RL with Libera (Fig. 10a), Libera+RL's average error rate is slightly higher than that of Libera's. To see the reason, we calculate the peak error rates of Libera+RL and Libera, which are 89 and 69%, respectively. In addition, among the 16 entity pairs, four and six entity pairs show non-zero error rates for Libera+RL and Libera. This means that, even if Libera+RL perfectly satisfies the *Min* of two more entity pairs than Libera, the error rate of an unsatisfied entity pair is much higher than that of Libera, which increases the average error rate. This is because of the rate limiting of Libera+RL; some entity pairs fully obtain the bandwidth of their *Min*, whereas the other entity pairs suffer from bandwidth starvation, which leads to high error rates (e.g., 89%). On the other hand, Libera does not control any bandwidth, so the traffic between entity pairs compete for the limited bandwidth. Thus, many of them do not satisfy *Min* requirements but also do not suffer from starvation. Therefore, compared with Libera+RL, Libera shows less value on the highest error rate.

In Fig. 10b of multi-tenant results, the peak error rates of Libera+RL and Libera are 96 and 60%, respectively, even with one tenant. Similar results are seen with two to four tenants. Note that TeaVisor shows 0% error rates with one to four tenants, which demonstrates that TeaVisor successfully guarantees *Min* under the varying number of tenants. The results are caused by the reasons explained in previous sections.

2) Maximum Bandwidth Requirements: We evaluate the effect of *Max* by varying the total amount of bandwidth requirements and the number of tenants. The amount of bandwidth higher than *Max* is considered as an error, and the *Max* error rate of zero means that the entity pair has reserved its bandwidth lower than *Max*. When the bandwidth is below *Min*, the maximum error rate becomes zero. The evaluation results are shown in Figs. 11 and 12.

Fig. 11a shows the results of 16 entity pairs. In [*Min, Min*], TeaVisor, Libera+RL, and Libera show average error rates of 5, 6, and 519%, respectively, over the total bandwidth requirements. Additionally, in Fig. 11b which illustrates [*Min, Max*], the error rates of TeaVisor, Libera+RL, and Libera are 0.3, 0, and 360%, respectively. We analyze the results as follows. First, the high error rates of Libera are because entity pairs consume bandwidth arbitrarily. Second, TeaVisor and Libera+RL show quite similar error rates on *Max*, meaning that both NHs provide bandwidth lower than *Max*. However, Libera+RL does not provide additional bandwidth for work conservation. For the evaluations of Fig. 11b where TeaVisor reserves additional bandwidth within *Max*, we find that entity pairs with TeaVisor gain 18.1% higher throughputs than *Min* on average (these results are not depicted because of space limitations). Considering that we set *Max* as 20% higher than *Min*, this result is reasonable. On the other hand, entity pairs in Libera+RL lose 9% throughput than their *Min*, implying less network utilization.

Next, Fig. 12a and 12b show the *Max* error rates when varying the number of tenants. For Fig. 12a, the average error rates of TeaVisor, Libera+RL, and Libera are 5.9, 5.9, and 280%, respectively. In addition, in Fig. 12b, their average error rates are 0, 0, and 323%, respectively. The reasons for such results are the same those shown in Fig. 11.

3) Throughput: We now explain the actual throughput that each entity pair gains with four tenants. The sum of *Min* given to this experiment is 75% of the entire network capacity. Fig. 13a and 13b show the bandwidth requirement combinations of [*Min, Min*] and [*Min, Max*], respectively. As expected, the throughput of TeaVisor is higher than *Min* but lower than *Max* (106% of *Min* in Fig. 13a and 119% of *Min* in Fig. 13b). On the other hand, Libera+RL shows 42% less throughput on average than *Min* for four entity pairs (4, 5, 12, and 13 in Fig.

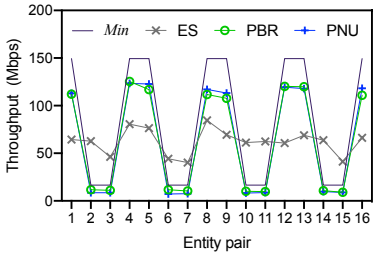


Fig. 14: Throughput changes of entity pairs per *Max* assurance policies.

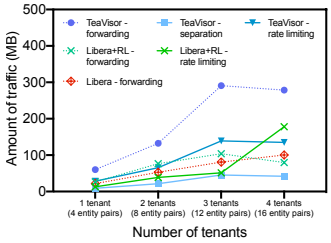


Fig. 15: Control traffic comparison.

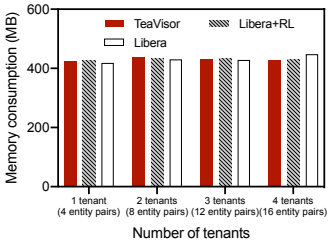


Fig. 16: Memory consumption comparison.

13a), and Libera also shows 47% less throughput than *Min* on four entity pairs (1, 4, 8, and 16 in Fig. 13a). In terms of *Max*, Libera+RL does not exceed *Max* similar to TeaVisor. However, for Libera, 10 entity pairs show higher throughput than *Max* (all entity pairs except 1, 4, 8, 9, 12, and 16 in Fig. 13b). The reasons for such results are similar to those explained in §IV-B1 and §IV-B2.

C. Effects of Max Assurance Policies

Fig. 14 shows the amount of additionally reserved bandwidth per *Max* assurance policies (i.e., ES, PBR, and PNU) explained in §III-C2. We investigate how the idle network capacity is divided for each entity pair. The experiment settings are explained in §IV-A2. The x-axis represents each entity pair, whereas the y-axis shows the additional bandwidth reserved by policies. The graphs in Fig. 14 are the throughput gained for each entity pair when the policy is applied. Eight entity pairs (i.e., 1, 4, 5, 8, 9, 12, 13, and 16) consume 90% of the total *Min* (consuming high bandwidth), and the other eight consume the remaining 10% (consuming low bandwidth). So, the graphs in Fig. 14 look like a step function. In addition, the line without marks is the amount of *Min*.

In Fig. 14, PBR and PNU have similar patterns. However, ES is different in that the additional throughput is less than PBR and PNU in entity pairs that consume high bandwidth and more with entity pairs consuming low bandwidth. On average, however, the sum of the additional throughput 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.

D. Overheads

1) Control Traffic Consumption: Fig. 15 shows the total amount of control traffic generated for installing forwarding, separation, and rate limiting rules. In addition to the forwarding rule, Libera+RL installs a rate limiting rule, and TeaVisor installs separation rule and rate limiting rules to satisfy the bandwidth isolation guarantee. 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. 15 shows that TeaVisor consumes 2.6and 2.9 times more control traffic than Libera+RL and Libera. This overhead is unavoidable for TeaVisor to provide substantial improvements on bandwidth isolation guarantee. However, considering that flow rule installations occur at once for a path between entity pairs, we believe that this overhead does not affect the throughput.

2) Additional Memory Consumption: Fig. 16 shows the average memory consumption. It is expected that TeaVisor consumes additional memory for the 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 studies without the bandwidth isolation guarantee.

V. Discussion

We discuss further research issues related to TeaVisor:

Scalability. As the SDN structure manages a network with a central entity, it could become a bottleneck in terms of scalability. This is similar to TeaVisor in that it provides the bandwidth isolation guarantee in NH. In fact, this problem is a fundamental challenge in the SDN structure that puts network control with a central entity. As a solution, several studies and SDN controllers suggest a physically distributed but logically central structure. This architecture has also been designed for NH [41]. Thus, we expect TeaVisor to be readily extended to such architectures.

Applicability on SDN. One might be concerned about whether TeaVisor can work with a non-virtualized SDN. We think this is possible because there is already an approach [6] to use SDN-NV for managing a physical network with existing SDN controllers. The components of TeaVisor could act as a proxy between the controllers and the physical network.

VI. 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 evaluation cases. We find that TeaVisor achieves near-zero error rates for both the minimum and maximum bandwidth requirements. In addition, through implementation optimization, TeaVisor consumes no additional memory compared with existing NHs.

We hope TeaVisor to trigger future research applying SDN-NV to real-world scenarios. We plan to work on predicting the control traffic in order to improve the accuracy of the prediction so that bandwidth isolation guarantee of TeaVisor can become practically meaningful.

REFERENCES

[1] A. Rodriguez-Natal, V. Ermagan, A. Noy, A. Sahai, G. Kaempfer, S. Barkai, F. Maino, and A. Cabellos-Aparicio, "Global state, local decisions: decentralized NFV for ISPs via enhanced SDN," *IEEE Communications Magazine*, vol. 55, no. 4, pp. 87–93, 2017.

[2] C. Janz, L. Ong, K. Sethuraman, and V. Shukla, "Emerging transport SDN architecture and use cases," *IEEE Communications Magazine*, vol. 54, no. 10, pp. 116–121, 2016.

[3] R. Sherwood, G. Gibb, K.-K. Yap, G. Appenzeller, M. Casado, N. McKeown, and G. M. Parulkar, "Can the production network be the testbed?" in *OSDI*, vol. 10, 2010, pp. 1–6.

[4] D. Drutskoy, E. Keller, and J. Rexford, "Scalable network virtualization in software-defined networks," *IEEE Internet Computing*, vol. 17, no. 2, pp. 20–27, 2012.

[5] A. Al-Shabibi, M. De Leenheer, M. Gerola, A. Koshibe, G. Parulkar, E. Salvadori, and B. Snow, "OpenVirteX: Make your virtual SDNs programmable," in *Proceedings of the third workshop on Hot topics in software defined networking*, 2014, pp. 25–30.

[6] X. Jin, J. Gossels, J. Rexford, and D. Walker, "CoVisor: A compositional hypervisor for software-defined networks," in *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*, 2015, pp. 87–101.

[7] 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.

[8] G. Yang, B.-y. Yu, H. Jin, and C. Yoo, "Libera for programmable network virtualization," *IEEE Communications Magazine*, vol. 58, no. 4, pp. 38–44, 2020.

[9] H. Jin, G. Yang, B.-y. Yu, and C. Yoo, "FAVE: Bandwidth-aware failover in virtualized SDN for clouds," in *2019 IEEE 12th International Conference on Cloud Computing (CLOUD)*, 2019, pp. 505–507.

[10] Y. Yoo, G. Yang, M. Kang, and C. Yoo, "Adaptive control channel traffic shaping for virtualized SDN in clouds," in *2020 IEEE 13th International Conference on Cloud Computing (CLOUD)*, 2020, pp. 22–24.

[11] P. Costa, M. Migliavacca, P. Pietzuch, and A. L. Wolf, "NaaS: Network-as-a-service in the cloud," in *2nd USENIX Workshop on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services (Hot-ICE 12)*, 2012.

[12] M. Kang, G. Yang, Y. Yoo, and C. Yoo, "TensorExpress: In-network communication scheduling for distributed deep learning," in *2020 IEEE 13th International Conference on Cloud Computing (CLOUD)*, 2020, pp. 25–27.

[13] J. Park, M. Naumov, P. Basu, S. Deng, A. Kalaiah, D. Khudia, J. Law, P. Malani, A. Malevich, S. Nadathur *et al.*, "Deep learning inference in facebook data centers: Characterization, performance optimizations and hardware implications," *arXiv preprint arXiv:1811.09886*, 2018.

[14] M. Kang, G. Yang, Y. Yoo, and C. Yoo, "Proactive congestion avoidance for distributed deep learning," *Sensors*, vol. 21, no. 1, 2021.

[15] M. Chowdhury, Z. Liu, A. Ghodsi, and I. Stoica, "Hug: Multi-resource fairness for correlated and elastic demands," in *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*, 2016, pp. 407–424.

[16] J. C. Mogul and L. Popa, "What we talk about when we talk about cloud network performance," *ACM SIGCOMM Computer Communication Review*, vol. 42, no. 5, pp. 44–48, 2012.

[17] L. Popa, G. Kumar, M. Chowdhury, A. Krishnamurthy, S. Ratnasamy, and I. Stoica, "FairCloud: Sharing the network in cloud computing," in *Proceedings of the ACM conference on SIGCOMM 2012*, 2012, pp. 187–198.

[18] 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 *Proceedings of the ACM conference on SIGCOMM 2013*, 2013, pp. 351–362.

[19] 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 Transactions on Cloud Computing*, 2018.

[20] F. Liu, J. Guo, X. Huang, and J. C. Lui, "eBA: Efficient bandwidth guarantee under traffic variability in datacenters," *IEEE/ACM Transactions on Networking*, vol. 25, no. 1, pp. 506–519, 2016.

[21] J. Son and R. Buyya, "Priority-aware VM allocation and network bandwidth provisioning in software-defined networking (SDN)-enabled clouds," *IEEE Transactions on Sustainable Computing*, vol. 4, no. 1, pp. 17–28, 2018.

[22] T. Lam, S. Radhakrishnan, A. Vahdat, and G. Varghese, *NetShare: Virtualizing data center networks across services*. [Department of Computer Science and Engineering], University of California . . . , 2010.

[23] K. Lee, C.-H. Hong, J. Hwang, and C. Yoo, "Dynamic network scheduling for virtual routers," *IEEE Systems Journal*, 2019.

[24] D. S. Marcon, F. M. Mazzola, and M. P. Barcellos, "Achieving minimum bandwidth guarantees and work-conservation in large-scale, SDN-based datacenter networks," *Computer Networks*, vol. 127, pp. 109–125, 2017.

[25] 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 *Proceedings of the 2018 Conference on Research in Adaptive and Convergent Systems*, 2018, pp. 80–87.

[26] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, A. Vahdat *et al.*, "Hedera: dynamic flow scheduling for data center networks." in *NSDI*, vol. 10, no. 8, 2010, pp. 89–92.

[27] M. Alizadeh, T. Edsall, S. Dharmapurikar, R. Vaidyanathan, K. Chu, A. Fingerhut, V. T. Lam, F. Matus, R. Pan, N. Yadav *et al.*, "CONGA: Distributed congestion-aware load balancing for datacenters," in *Proceedings of the 2014 ACM conference on SIGCOMM*, 2014, pp. 503–514.

[28] N. Katta, A. Ghag, M. Hira, I. Keslassy, A. Bergman, C. Kim, and J. Rexford, "Clove: Congestion-aware load balancing at the virtual edge," in *Proceedings of the 13th International Conference on emerging Networking EXperiments and Technologies*, 2017, pp. 323–335.

[29] S. Ghorbani, Z. Yang, P. B. Godfrey, Y. Ganjali, and A. Firoozshahian, "Drill: Micro load balancing for low-latency data center networks," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, 2017, pp. 225–238.

[30] J. Guo, F. Liu, T. Wang, and J. C. Lui, "Pricing intra-datacenter networks with over-committed bandwidth guarantee," in *2017 USENIX Annual Technical Conference (USENIX ATC 17)*, 2017, pp. 69–81.

[31] A. Khan, A. Zugenmaier, D. Jurca, and W. Kellerer, "Network virtualization: a hypervisor for the internet?" *IEEE Communications Magazine*, vol. 50, no. 1, pp. 136–143, 2012.

[32] A. Bianco, P. Giaccone, R. Mashayekhi, M. Ullio, and V. Vercellone, "Scalability of ONOS reactive forwarding applications in isp networks," *Computer Communications*, vol. 102, pp. 130–138, 2017.

[33] W. Bai, L. Chen, K. Chen, D. Han, C. Tian, and H. Wang, "PIAS: practical information-agnostic flow scheduling for commodity data centers," *IEEE/ACM Transactions on Networking*, vol. 25, no. 4, pp. 1954–1967, 2017.

[34] J. Mudigonda, P. Yalagandula, J. Mogul, B. Stiekes, and Y. Pouffary, "NetLord: a scalable multi-tenant network architecture for virtualized datacenters," *ACM SIGCOMM Computer Communication Review*, vol. 41, no. 4, pp. 62–73, 2011.

[35] G. Yang, B.-y. Yu, W. Jeong, and C. Yoo, "FlowVirt: Flow rule virtualization for dynamic scalability of programmable network virtualization," in *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*. IEEE, 2018, pp. 350–358.

[36] G. Yang, H. Jin, M. Kang, G. J. Moon, and C. Yoo, "Network monitoring for SDN virtual networks," in *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*. IEEE, 2020, pp. 1261–1270.

[37] A. Fischer, J. F. Botero, M. T. Beck, H. De Meer, and X. Hesselbach, "Virtual network embedding: A survey," *IEEE Communications Surveys & Tutorials*, vol. 15, no. 4, pp. 1888–1906, 2013.

[38] B. Pfaff, J. Pettit, T. Koponen, E. Jackson, A. Zhou, J. Rajahalme, J. Gross, A. Wang, J. Stringer, P. Shelar *et al.*, "The design and implementation of open vswitch," in *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*, 2015, pp. 117–130.

[39] P. Berde, M. Gerola, J. Hart, Y. Higuchi, M. Kobayashi, T. Koide, B. Lantz, B. O'Connor, P. Radoslavov, W. Snow *et al.*, "ONOS: towards an open, distributed sdn os," in *Proceedings of the third workshop on Hot topics in software defined networking*, 2014, pp. 1–6.

[40] V. Gueant, "iperf-the tcp, udp and sctp network bandwidth measurement tool," *Iperf. fr. Np*, 2017.

[41] Y. Han, T. Vachuska, A. Al-Shabibi, J. Li, H. Huang, W. Snow, and J. W.-K. Hong, "ONVisor: Towards a scalable and flexible sdn-based network virtualization platform on ONOS," *International Journal of Network Management*, vol. 28, no. 2, p. e2012, 2018.