Distributed and Efficient Network Hypervisor for SDN Virtualization

Ling Xia Liao[1,2], Jian Wang[3], Han-Chieh Chao[2], Bin Qin[4]

[1] School of Electronic Information and Automation, Guilin University of Aerospace Technology, China
[2] Department of Electrical Engineering, National Dong Hwa University, Taiwan
[3] School of Information and Communication, Guilin University of Electronic Technology, China
[4] Information Center, Guilin University of Aerospace Technology, China
Liaolx@guat.edu.cn, Wangjian@guet.edu.cn, hcc@gms.ndhu.edu.tw, qinbin@guat.edu.cn

Abstract

This article proposes a novel Distributed and Efficient Network Hypervisor (DeNH) to virtualize Software Defined Networking systems. DeNH adopts container-based architecture to reduce the extra delay imposed by the hypervisor, and provides efficient inter-hypervisor coordination to support global network embedding and global network addressing and control logic mapping table, leading to larger network scalability and higher network performance. A lightweight caching system with a record pre-fetching mechanism is proposed to significantly increase the cache hit rate for a further delay reduction in setting up flow entries. DeNH proposes to use the packets forwarded to the hypervisors in the control plane for flow setup to generate global network statistics. Such strategy significantly reduces the overhead imposed by shipping statistics from data to control planes. Evaluations over emulated SDN systems demonstrate the extra flow setup delay added by DeNH is very short. While the cache hit rate of DeNH is estimated to 100% using the traces of real data centres, the statistics maintained by DeNH can give an accurate detection of elephant flows in networks.

Keywords: Distributed network hypervisor, Virtualization, SDN, Cache, Detection of elephant flows

1 Introduction

Network virtualization and Software Defined Networking (SDN) [1] are two of the key technologies for complicated systems to achieve efficient and intelligent resource sharing and management [2]. As similar as computer hypervisors allocating resources to and manage virtual machines in computer virtualization, network hypervisors (NHs) allocate resources to and manage the operation of virtual networks (VNs) in network virtualization [3]. While network resources may be virtualized at different networking layers [4], NHs can virtualize networks across layers to form fully virtualized networks [5].

Fully virtualized networks have VNs managed by their own controllers with their own network resources [3]. They are highly demanded by current data centres, as they allow multiple VNs to share the same physical infrastructure [3]. However, employing an NH to fully virtualize a conventional computer and communication network (CCN) is challenging, because that need the NH to coordinate with all the devices in the network. However, enabling such coordination over a conventional CCN is difficult due to the lack of mechanisms to flexibly configure and manage the wide range of networking devices. As SDN systems have control planes separated from networking devices, with standardized protocols and interfaces [6] to configure and program the devices, SDN control planes make the perfect place for launching NHs.

Current research has proposed some NHs over SDN systems [3]. No matter such solutions are centralized or distributed (centralized solutions utilize a single NH entity in the control plane, distributed solutions can have multiple NH entities), each packet of flows forwarded to tenant controllers for the installation of flow entries over VNs must go through the NH entity for the conversion of control logic and network addresses between virtual and physical networks (PNs), leading to an extra delay imposed by the NH entities [7]. Tenant controllers manage VNs over a PN. Having a low flow setup delay is crucial for a better quality of service and user experience over virtualized SDN systems.

A centralized compact design that enables one container-level NH entity and multiple tenant controllers can reduce such delays, because container-level NH and tenant controller entities in a compact design are application modules sharing the same operating system and physical host [5], and their interaction is simplified. However, having the NH and tenant controller modules residing in the same host

*Corresponding Author: Han-Chieh Chao; E-mail: hcc@gms.ndhu.edu.tw
DOI: 10.3966/160792642021052203012

limits the number of tenant controller modules a host can launch, and hence the number of VNs an NH entity can manage, given each VN has its own tenant controller module [3]. The number of VNs a PN can support is a metric to measure the scalability of an NH solution [1]. To address this issue, distributed and compact designs with multiple hypervisor entities, as illustrated in Figure 1, can be applied [5].

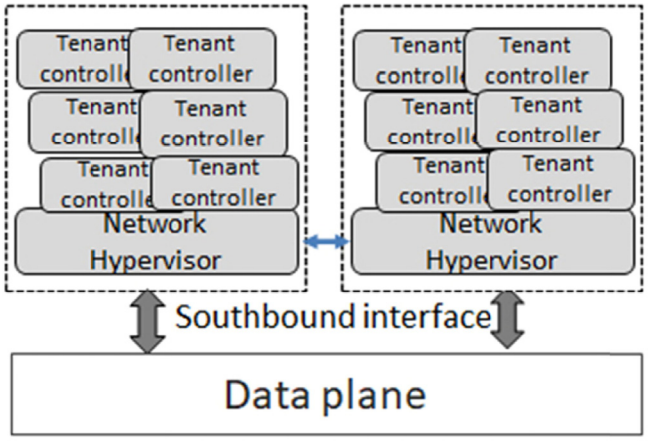


Figure 1. Distributed and compact solution with container-level NH and tenant controllers

However, such designs often lack inter-hypervisor coordination [5, 8-10], causing a difficulty in assembling global network view and enabling global network embedding, leading to limited network scalability. The lack of inter-hypervisor coordination also makes the maintenance of the global mapping table very challenging. Global mapping tables consist of mapping records that convert the control logic and network addresses for all the VNs at each hypervisor entity. Maintaining a global mapping table at an NH entity is crucial as it allows an NH entity to quickly take over the work of the other NH entities when they are overloaded or in failures.

Previously proposed distributed NHs often implemented a read caching system caching the mapping records at an NH entity to reduce the extra delay imposed by NH entities, but that reduction is trivial. The cache system often loads a mapping record from the original storage when the first packet of a flow is forwarded to the control plane. However, the cached mapping record may not be visited again, because many flows in the networks are short flows and they only forward their first packets to the control plane, leading to a cache hit rate of 0 [5, 8-9].

Previously proposed distributed NHs also lack efficient approaches to collect global network statistics to support intelligent resource management [9, 11-14]. Network statistics often have big volumes and must be shipped from switches to controllers in SDN systems. Shipping network statistics adds extra overhead on both control and data planes, reducing the network performance and scalability [8].

To address these issues, we adopt the distributed and compact design and make the following major contributions in this article:

(1) A distributed and efficient network hypervisor (DeNH) with container-level hypervisor and tenant controllers, is proposed by extending our previous proposed distributed controller (DisCon) [15]. DeNH maintains a global network topology and global mapping table for a larger scalability.

(2) A lightweight mapping table cache with a record pre-fetching mechanism is proposed to effectively reduce the flow setup latency over VNs.

(3) Flow statistics is collected based on the packets of flows forwarded to the control plane to significantly reduce the overhead of both control and data planes.

(4) Extensive evaluations are provided to demonstrate that the scalability of DeNH can be enhanced by using multiple hypervisor entities, although the physical resources of a computer hosting a DeNH entity still limits the scalability of DeNH. As the mapping table cache of DeNH has a cache hit rate of 100%, the extra delay imposed by the NH is very small. The proposed flow statistics demonstrates a high accuracy in detecting elephant flows that often last a long period with many packets to heavily impact the network performance [16].

The rest of this article is organized as the follows. Section 2 summarizes the related work and background. DeNH is proposed and prototyped in Sections 3 and 4, respectively. Evaluations are given in Section 5, and conclusions are drawn in Section 6.

2 Background and Related Work

2.1 Mechanisms for Network Hypervisors

Network virtualization should provide resource abstraction and isolation for VNs. PNs can be virtualized in widely various levels. The lowest level provides a transparent one-to-one mapping in which virtual topologies are constructed over the physical topology, while the highest level virtualizes the entire physical topology as a single virtual link or node, and generally, there is a wide range of levels between of them [3].

NHs are VN monitors that can fully virtualize SDN systems and provide various levels of abstraction and isolation [5, 17]. At the data plane, NHs can employ Link Layer Discovery Protocol (LLDP) [18] to abstract network topology, VLAN [19] and NVGRE [20] protocols to abstract link capabilities, and OpenFlow protocols [6] to slice port queues of switches. At the control plane, NHs assign a unique ID to a slice of links [19-20] or a VN [5]. VNs may have distinct segments of PN addressing space or overlapped network addressing spaces [5]. NHs can allow each VN to have its own tenant controller [1, 9, 11-12, 21] or container-level tenant controller [5] to isolate the control logic from other VNs.

2.2 Hypervisors Over SDN Systems

FlowVisor [21] is the first centralized NH solution that uses OpenFlow protocols to virtualize SDN systems. It introduces the flowspace and allocates a VN its own flowspace to differentiate from the other VNs. However, it does not hide physical switches and network addresses in VNs. AdVisor [11], VeRTIGO [12], Enhanced FlowVisor [17], and OpenVirteX [8] are centralized NHs extending FlowVisor in different ways. OpenVirteX can hide physical switches and network addresses in VNs by providing a mapping table to convert control logic and network addresses between PNs and VNs, while the other three cannot. There are also centralized NHs not based on FlowVisor. They may focus on the isolation between virtual slices sharing SDN switches [2] or target specific networking scenarios such as wireless and mobile networks [1].

While DFVisor [10] is a distributed NH solution based on FlowVisor and does not hide the physical switches and network addresses for VNs, some non-FlowVisor based distributed NHs such as NVP [13], FlowN [5], ONVisor [9], and AutoSlice [14] do. FlowN does it by converting the control logic and network addresses between VNs and PNs using a mapping table. It reduces the flow setup latency over VNs by employing a compact design with container-level hypervisors and tenant controllers. It maintains global mapping table by replicating data among databases. It provides no inter-hypervisor coordination to abstract network topology and embed VNs globally, but NVP, ONVisor, and our proposed DeNH do by inheriting it form some distributed controller solutions. For instance, NVP wraps Onix [22] controller, ONVisor inherits the distributed design from ONOS [23], and the proposed DeNH extends DisCon. While ONVisor allows each VN to be managed by an individual controller, NVP and DeNH do not. NVP acts as a controller to manage the application and flows over VNs, while ONVisor uses a distributed data store to store and share the global network view among multiple nodes. DeNH has a distributed and compact design similar as FlowN but provides a more efficient inter-hypervisor coordination. While NVP, ONVisor, and AutoSlice focus on developing novel mechanism for resource abstraction and isolation, DeNH makes great efforts on reducing the flow setup latency of VNs and the overhead of statistics collection.

2.3 Caches Over SDN Systems

SDN controller solutions often implement read caches to avoid directly operating the original global network view in the databases or DSs [22-23]. Read caches load a new record from the original storage when a cache miss is generated. Hypervisor solutions such as FlowN, NVP, ONVisor, AutoSlice, and OpenVirteX also implemented read caches to cache mapping table records. However, they often suffer a low cache hit rate in SDN systems, in which the first packets of new flows are forwarded to the control plane for the installation of the flow entries. In NH solutions, such first packets need to visit the mapping cache to convert their control logic and network addresses. Since they are the first packets of the new flows, they suffer a cache miss and load new mapping records from the original mapping table to the cache. This allows the following packets of the flow forwarded to the control plane to directly retrieve the mapping record without involving the original mapping table again. Since many flows in current networks are mice flows that only forward their first packets to the control plane [24], many mapping records loaded by the first packets will never be read again, leading to a high cache miss rate. Such a high cache miss rate increases the flow setup latency, and cannot be decreased by increasing the size of the cache. DeNH addresses this issue by developing a mapping records pre-fetching mechanism.

2.4 Statistics Over SDN Systems

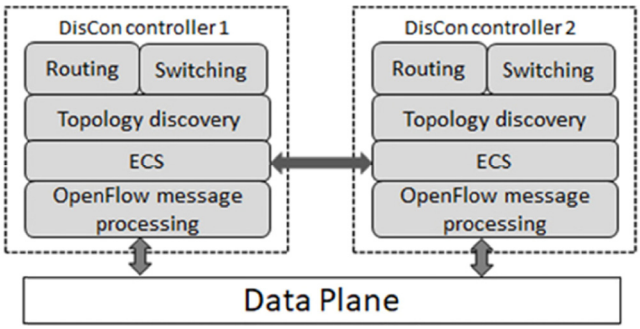
Network statistics are a major part of the global network view. While conventional CCNs often deploy traffic analysis tools to sample network packets and generate the network statistics based on the samples, SDN systems can count flow statistics at switches without sampling. SDN systems may employ active or passive methods to collect statistics. Active methods often let the conrollers to repeatedly poll the network statistics from switches, while passive methods force switches to report their statistics automatically. Since the flow statistics generated by both types of methods are counted by all flow entries at switches, shipping flow statistics from switches to controllers are unavoidable. This imposes extra overhead on both control and data planes [16]. DeNH addresses this issue by counting the flow statistics using the packets of flows forwarded to the control plane instead of shipping statistics from switches to controllers.

3 Distributed and Efficient Network Hypervisor

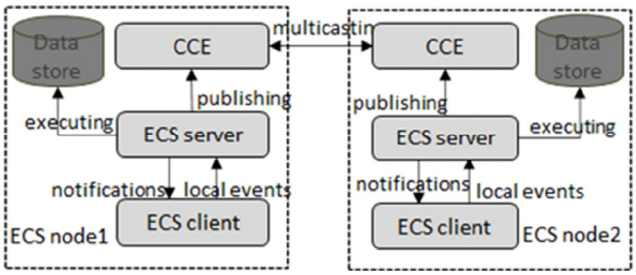
3.1 DisCon's Architecture and Major Mechanisms

Our previous work, DisCon is a distributed controller with multiple controller entities (nodes), as shown in Figure 2(a). It proposed a novel event coordination system (ECS) to enable the inter-controller coordination. As shown in Figure 2(b), ECS consists of multiple nodes, each containing an ECS server and multiple clients. An ECS server contains three processes, the server, the in-memory data store (DS), and the Corosync Cluster Engine (CCE) [25], running on the same host. An ECS server should run

independently. An ECS client should be integrated into the local controlling entities to operate the local DS via the local ECS server. ECS uses CCE to fully replicate the local events among all the node.



(a) Architecture of DisCon



(b) Architecture of ECS

Figure 2. Architecture of DisCon and ECS

3.2 DeNH's Architecture and Functions

DeNH is a distributed NH solution extended from DisCon for the virtualization of SDN systems. Compared with DisCon consisting of multiple controllers, each managing a subset of switches of an SDN system, DeNH consists of multiple hypervisor entities, each managing a subset of VNs over the PN. Particularly, as shown in Figure 3(a), DeNH inserts an NH module into a DisCon entity and allows a VN to have its own tenant controller module.

Given a VN with the requirements on network topology and resources, the NH module at a DeNH node can embed it onto the PN using any existing network embedding algorithm. DeNH uses a 4-byte integer (the tenant ID) to identify a VN. After a VN is successfully embedded, a tenant ID is assigned. As DisCon can construct the global network topology at each controller using the ECS, each DeNH entity (node) can abstract the global network topology and embed a VN globally.

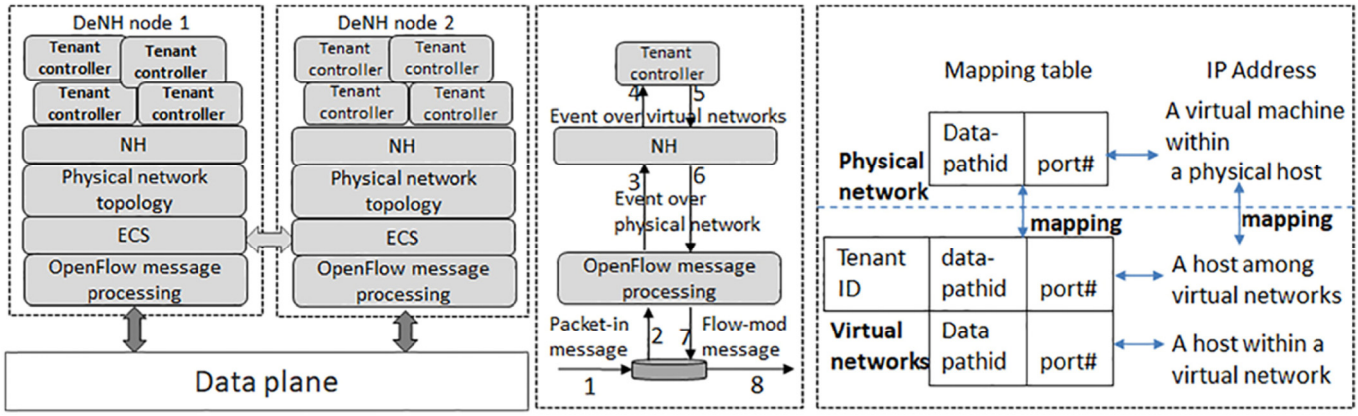
Ports are the basic resources of switches. Current OpenFlow protocols define logical ports and allow to allocate resources (such as bandwidth) to them. DeNH uses logical ports to abstract and isolate the resource of switches and links at the data plane. Particularly, DeNH launches a virtual machine for each VN connecting to a logical port of a switch using a one-to-one match system, as shown in Figure 3(c). The virtual machine with physical IP address over the PN can be identified by the combination of the logical port (identified by the port #) and the physical switch (identified by its datapathid) that the virtual machine connects to. Let a VN have a tenant ID, the combination of the virtual switch (identified by the tenant datapathid) and its port (identified by the tenant port#) that connects the virtual machine to the VN identifies a host (the virtual machine with tenant IP address) within the VN. The combination of the tenant ID, the tenant datapathid, and the tenant port# identifies a host among all the VNs over the given PN. This matching system allows VNs to have overlapped network addresses and facilitates the conversion of network addresses between VNs and PNs, while providing isolation at both control and data planes.

At the control plane, although DeNH allows each VN to have its own tenant controller module, the NH and OpenFlow message processing modules at a node are shared by VNs managed by the node. However, we can avoid a VN affecting the performance of the other VNs by setting the maximal number of OpenFlow messages a VN can process.

Figure 3(b) illustrates the workflow of DeNH. Particularly, when running applications over a VN, packets of flows are generated by the applications and forwarded by the involved switches (1). If such switches set up flows reactively, the first packet of each flow will be packed to a packet-in message and forwarded to the control plane for the installation of a flow entry (2).

The packet-in message will be received by the OpenFlow message processing module at a DeNH entity, and further packed to an event and sent to the NH module (3). The NH module converts the received event from the physical network to virtual networks and sends it to their corresponding tenant controller module for processing (4). The corresponding tenant controller module receives the event, generates the response, and sends it back to the NH module (5), in which the response is converted from the VN to the physical network and sent to the OpenFlow message processing module (6) again. The OpenFlow message processing module further packs the response to an OpenFlow message and forwards it back to the corresponding switch (7). The switch processes the received message, installs the corresponding flow entry, and forwards the received packet based on the installed flow entries (8). In (4) and (5), the NH module relies on the mapping table to convert the events between VNs and PNs.

The mapping table in DeNH is generated based on the one-to-one matching system and written to the DS of an ECS server. Since the ECS server can replicate its write operations to other ECS servers, the update of a mapping table at a DeNH entity can be synchronized to the other mapping tables, such that a global mapping



(a) Architecture (b) workflow (c) one-to-one matching system

Figure 3. DeNH's architecture, work flow, and matching system

table consisting of the mapping records of all the virtual networks can be maintained at each DeNH node. In the case that some DeNH nodes are overloaded or in failures, maintaining a global mapping table at each DeNH node allows a DeNH node to quickly take over the works of the other DeNH nodes without replicating the mapping records from the other mapping tables to improve the network availability.

3.3 Performance and Scalability Enhancement

To enhance the whole system's performance and scalability, DeNH uses the distributed and compact design with inter-hypervisor coordination facilitated by ECS. DeNH splits its VNs into two groups: the VNs having particular concerns on control logic and security and the VNs having no such concern. A default tenant controller module is provided to manage the former group. DeNH also develops the following mechanisms for the enhancement.

3.3.1 Mapping Table Cache with Record Pre-fetching

NH solutions over SDN systems often have a low cache hit rate, leading to a longer network setup latency that decreases the network performance. DeNH proposes a lightweight read cache system with record pre-fetching mechanism. DeNH stores its mapping table in the DS of ECS. As listed in Table 1, DeNH classifies the mapping records as two categories: Cat1 records that convert flows from a VN to the PN, and Cat2 records that convert flows from the PN to a VN. Each record in the cache has the same key and value as that in the original mapping table.

Table 1. Mapping table of DeNH

Type	Key	Value	For
Cat1	tenant ID, ten-ant datapathid, tenant port #	Datapathid, port#	Virtual-physical
Cat2	Cat1's Value	Cat1's Key	Physical-virtual

Since the NH module always maintains a list of tenant IDs identifying the VNs managed, and all the Cat1 records in the mapping table place tenant IDs in their keys, the NH module can integrate an ECS client to register the Cat1 records that have tenant IDs in the list. When these Cat1 records are established or updated in the DS, the involved write operations are notified to the ECS client.

Therefore, the NH module that integrates the ECS client receive the write operations and synchronize the updates to its mapping table cache. This way, all the Cat1 records are loaded to the cache without triggered by cache misses. Since every Cat1 record has a corresponding Cat2 record with the key and value switched in the mapping table, the corresponding Cat2 record also can be synchronized to the cache.

When mapping records need to be updated, the updates are always made over the original mapping table via ECS write operations. These updates are then synchronized to the cache using the procedure discussed above. Figure 4 shows this synchronization process between the original mapping table and its cache within a DeNH entity. Since ECS can replicate write operations among multiple ECS servers, the mapping records written to a mapping table are also synchronized to the other mapping tables located in other DeNH entities. This synchronization among multiple mapping tables, as also shown in Figure 4, enables each DeNH entity to have its own global mapping table such that a DeNH entity can quickly take over the work of the other DeNH entities for load balancing or higher network availability.

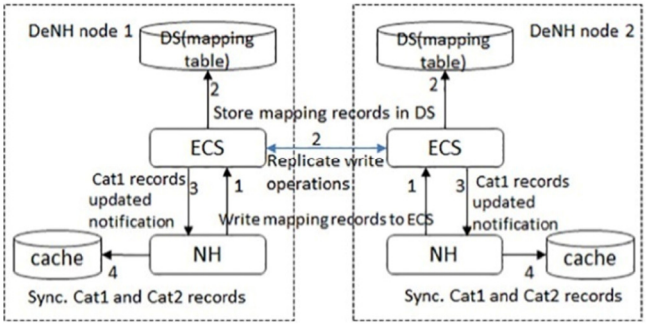


Figure 4. Synchronization mapping tables and the cache

3.3.2 Flow Statistics

Since the packets of flows forwarded to the control plane must firstly retrieve mapping record from the mapping table cache to convert their control logic and network addresses, the mapping table cache can count the packets forwarded to the control plane. Particularly, DeNH lets the cached mapping record *i* count the total number of times (c_i), the very first time ($t_{i-first}$), and the most recent time ($t_{i-recent}$) it has been read, and the total bytes of the packets (p_i) that have visited it. The statistics are attached to the end of each cached mapping record.

SDN switches only forward the packets of flows to the control plane when the matched flow entries are not found in their flow tables [6]. There are two scenarios in which the matched flow entries may not be found: 1) the flow entries have not been installed and 2) the installed flow entries have timed out. The first scenario makes the first packet of a flow forwarded to the control plane, and the second scenario determines if one of the following packets of a flow should be forwarded to the control plane.

Given a measurement period *T*, let *F* be the set of flows passing through the network. Let flow $f_i \in F$ consist of a sequence of packets, p_{ij} be the *j*th packet of f_i, t_{ij} be the arriving time of p_{ij}, and P_i be the total number of packets of f_i. The total number of packets of flows passing through the network (P) during the *T* is computed by equation (1), and the total number of packets forwarded to the mapping table cache (P_c) is calculated by equation (2), where δ_i is the number of packets of f_i returning the control plane caused by the timed-out flow entry.

$$P = \sum_{f_i \in F} P_i \tag{1}$$

$$P_c = \sum_{f_i \in F} (1 + \delta_i) \tag{2}$$

$$\delta_i = \sum_{p_{ij}, p_i(j-1), p_{ik} \in f_i, k<j, j>1} a_{ij} \tag{3}$$

$$a_{ij} = \begin{cases} 1, if\ t_{ij} - t_{ik} \geq t_{hard} \\ 0, otherwise \end{cases} \text{ and } k = \begin{cases} j, if\ a_{ij} = 1 \\ k, otherwise \end{cases} \tag{4}$$

$$a_{ij} = \begin{cases} 1, if\ t_{ij} - t_{i(j-1)} \geq t_{idle} \\ 0, otherwise \end{cases} \tag{5}$$

$$A = \sum_{f_i \in F} (1 + \delta_i) / P \tag{6}$$

To compute δ_i, we consider two scenarios: the flow entries with hard and idle timeouts. For each $p_i \in f_i$ we define a binary variable a_{ij}. We let $a_{ij} = 1$ if p_{ij} returns to the control plane, and $a_{ij} = 0$ otherwise. In the case of flow entries with hard timeout t_{hard}, let p_{ik} renew the flow entry ($k < j$), and the packets between p_{ik} and p_{ij} haven't renewed the flow entry. Then, we let $a_{ij} = 1$ and $k = j$ if $t_{ij} - t_{ik} \geq t_{hard}$; and $a_{ij} = 0$ otherwise, as shown in equation (4). This means that p_{ij} will be forwarded to the control plane if at its arriving time the flow entry has stayed in the flow table longer than the given hard timeout. In the case of flow entries with idle timeout t_{idle}, we compute the time difference between $t_{ij} - t_{i(j-1)}$ and compare it with t_{idle}. We let $a_{ij} = 1$ if $t_{ij} - t_{i(j-1)} \geq t_{idle}$, and $a_{ij} = 0$ otherwise, as shown in equation (5), which means that p_{ij} will be forwarded to the control plane if the idle time of the flow entry is not less than t_{idle}. Therefore, the parameter *A* that represents the percentage of the total number of packets of flows counted by the statistics can be computed by eqation (6). All the variables used in the equations are listed in Table 2.

4 DeNH Prototyping

Current DeNH implementation consists of an NH module developed from scratch, and an OpenFlow message processing module, a topology discovery module, a default tenant switching module (TSM), and a default tenant routing module (TRM) respectively extended from DisCon.

The NH module incorporates the functions of embedding VNs, generating mapping records, converting network addresses, and converting control logics. A VN can be embedded by a DeNH entity in a global manner. After a VN is embedded successfully, its mapping records are generated based on the matching system and written to the mapping table that is stored in the DS of ECS. The function of converting

Table 2. Symbols and their descriptions

Symbols	Descriptions
T	the measurement period
F	the set of flows during T
P	the total number of packets of flows in F
A	the percentage of total number of packets of flows sent to control plane
c_i	the number of times the cached record i has been visited
$t_{i-first}$	the very first time when the cached record i has been visited
$t_{i-recent}$	the most recent time when the cached record i has been visited
p_i	the total bytes of packets that have visited the cached record i
f_i	the flow i in F
P_i	the total number of packets in f_i
P_c	the total number of packets of flows sent to the control plane in T
p_{ij}	the jth packet of f_i
t_{ij}	the arriving time of p_{ij}
t_{ik}	the time when the flow entry with hard timeout of f_i is set up
δ_i	the number of returning packets of f_i
a_{ij}	the binary integer showing if p_{ij} is forwarded to the control plane
t_{hard}	the hard timeout value
t_{idle}	the idle timeout value

network addresses translates network addresses between virtual and physical networks. The function of converting control logic translates the events between tenant controller and OpenFlow message processing modules. The NH module uses a hashmap to implement a mapping table cache. The Flow statistics are attached to the end of each cached mapping record.

The OpenFlow message processing module at a DeNH entity handles the OpenFlow messages of that entity, and it is shared by all the tenant controller modules of that entity. The topology discovery module discovers the physical topology over a network partition and constructs the global physical topology over the entire network partitions. To discover the topology of VNs, for each received topology event, the NH module is called to convert the received event from the physical network to the corresponding VN. The network topology is stored in a data structure (DpInfo) proposed by NOX [26]. DeNH allows each VN to deploy its own tenant controller module or use the default TSM and TRM, which implement a general flow switching procedure and a routing procedure [26] over VNs, respectively.

5 Evaluation

5.1 Mapping Table Synchronization

As each write operation issued by a DeNH entity is replicated to the other DeNH entities in the control plane, the mapping records written to one mapping table is synchronized to the other mapping tables of the other DeNH entities, leading to a synchronization delay. We simply use the half Round Trip Time (RTT) between the two DeNH entities to estimate the synchronization delay between two mapping tables as the write operations are multicasted by ECS servers. The half RTT in our test bed in a 1000M local area network is about 0.1ms.

5.2 Mapping Table Cache Synchronization

A DeNH entity can synchronize its original mapping table to the mapping table cache using the ECS. We measure the time difference between the time sending a write operation that updates a mapping record to its original mapping table and the time receiving the notification of the write operation such that the mapping table cache can be updated accordingly at a DeNH entity, to estimate the cache synchronization delay. We do not consider the half RTT to be the part of the cache synchronization delay, because the original mapping table and its mapping table cache are located on the same host. We let each write operation update 1-Kbyte data since the volume of each mapping record in the mapping table is smaller than 1-Kbyte. We let each such write request have a timestamp to record the time when the request is sent. We calculate the time difference between the time that the request sent and the notification received. We also calculate the delay in updating a mapping record to its original mapping table, using the synchronization write delay

of ECS when each write operating 1-Kbyte data. The results are shown in Figure 5.

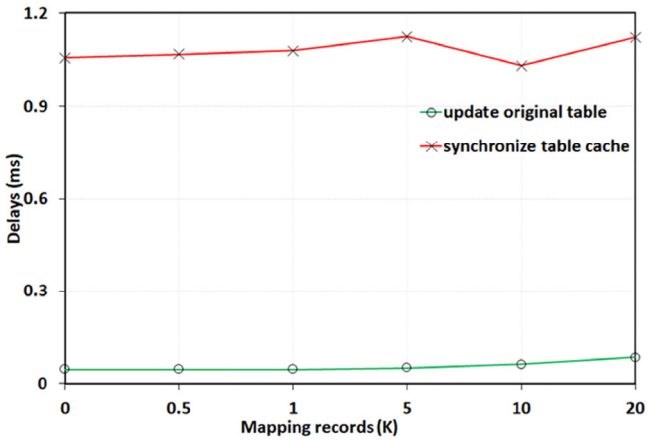


Figure 5. Original mapping table updating delay and mapping table cache synchronization delay

Apparently, the two delays do not increase significantly as more mapping records are included in a mapping table. This implies the delay imposed by a longer watch list that collects all the mapping records registered by a DeNH entity is not significant regarding the synchronization delay between a mapping table and its cache and the synchronization delay between two mapping tables. The cache synchronization delay remains 1.031 ms to 1.124 ms when the number of registered mapping records increases from 0 to 20K. Since a DeNH entity needs to register the Cat1 records for those VNs whose tenant controller modules are launched on the same host that hosts the DeNH entity, the number of registered mapping records implies the number of mapping records belonging to the supervised VNs, and hence the number of VNs a DeNH entity can supervise. In this sense, increasing the number of VNs supervised by a DeNH entity does not increase the cache synchronization delay and the mapping table synchronization delay.

5.3 Flow Setup Latency

To measure the flow setup latency of DeNH over VNs and have a clear comparison with some previously proposed solutions, we construct a test scenario similar as the ones described in [5] and [8].

Particularly, as listed in Table 3, we use a laptop to generate two virtual machines: one has three processors and 2GB memory to run a hypervisor entity, and the other has one processor and 2GB memory to run Cbench [27]. We let each physical port of a switch split up to 100 logical ports and construct 100 virtual networks that have the exact topology copy of the physical network as used in FlowN [5] and OpenVirteX [8]. We manually generate mapping tables, and we run Cbench to calculate the average latency of a switch from sending a packet-in message to receiving its response. This latency is the flow setup latency of DeNH. We let DeNH retrieve mapping records from its mapping table cache and compare its flow setup latency to those of FlowN, OpenVirteX, FlowVisor, and a non-virtualized base line (NOX), as shown in Figure 6, where FlowN and OpenVirteX also retrieve mapping records from their mapping table caches.

Table 3. Test setting for flow setup latency comparison

Solution	Testbed	**Descriptions**	**VN**
DeNH and NOX	laptop (i5-2700 CPU, 3.3G clock speed)	one virtual machine for hypervisor	100
FlowN &FlowVisor	laptop (i5-2700 CPU, 3.3G clock speed)	entity, one virtual machine for Cbench,	100
OpenVirteX	laptop (i7-3500 CPU, 2.7G clock speed)	same topology for physical and VNs	100

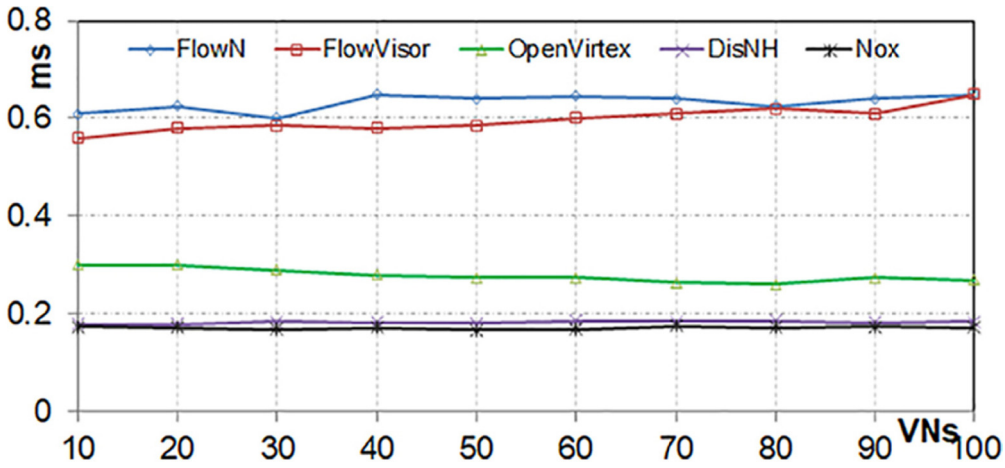


Figure 6. Flow setup latency Comparison

It is apparent that the flow setup latency of the listed hypervisors does not significantly change as the number of VNs increases. DeNH has much shorter flow setup latency than FlowN, FlowVisor, and OpenVirteX. The overall increase of the flow setup latency in DeNH against to NOX is less than 0.015 ms. We cannot compare DeNH' flow setup latency to ONVisor's [9, 28] and NVP's [13], since they are not open sourced, and their flow setup latencies have not been published.

We also compare the flow setup rate of a DeNH entity with a NOX controller. We emulate a linear network with 100 switches and 100 hosts, and we deploy a NOX controller (activating the switching module) to manage it. We generate 60 VNs with the same topology as the PNs and manage such a network using a DeNH entity respectively running a TSM and a TRM. We use Cbench to measure the flow setup rate of the control plane in these two scenarios. The results show that NOX can set up around 40K flows per second, DeNH running the TSM can set up about 35K flows per second, and DeNH running the TRM can set up about 17K flows per second. This implies that adding a hypervisor module between switches and tenant controllers reduces the control plane's flow setup rate due to the increased flow setup latency. Using TRM to set up flows has a lower flow setup rate than using TSM, because TRM must maintain global network topology while TSM does not. Maintaining global network topology adds overhead on the control plane, resulting a reduced flow setup rate.

5.4 Cache Hit Rate

Figure 6 was obtained by measuring the flow setup latency of FlowN and OpenVirteX when they have a 100% cache hit rate. However, FlowN and OpenVirteX cannot provide such a high cache hit rate in practice. To estimate it, we consider a packet trace that lasts 5 minutes and consists of 1,005,395 packets [28]. We let the packets with the same source and destination addresses constitute a flow and identify 3562 flows in the trace. We allow the flow entries to have idle and hard timeouts, and varying the value of timeouts from 0.05 to 0.1, 1, 5, and 10s. As a base line we count the number of packets of each flow in the trace. We count the number of packets of a flow in the trace being sent to the control plane under a certain timeout. This number is given by $1+\delta_i$ for f_i by (2), where δ_i is the number of returning packets of f_i. Therefore, the cache hit rate of FlowN and OpenVirteX can be calculated by $\delta_i(1+\delta_i)$, since the first packet forwarded to the control plane suffers a cache miss that loads the mapping record to the cache; and the cache hit rate of DeNH is given by $(1+\delta_i)/(1+\delta_i) = 100\%$, since all the mapping records have been pre-loaded.

It is apparent that each flow at least sent one packet (the first packet of a flow) to the control plane, and the number of packets that a flow sent to the control plane was significantly smaller than the total packets of the flow, especially for those flows with many packets. Particularly, as listed in Table 4, 17.7%, 18.3%, 25.5%, 33.9%, and 38% of the flows only send their first packets to the control plane, if flow entries have idle timeout that is set as 0.05, 0.1, 1, 5, and 10 s, respectively. If flow entries have hard timeout set as 0.05, 0.1, 1, 5s, and 10s, such percentage increases to 18.7%, 19.5%, 30.2%, 44.5%, and 51.6%, respectively. This implies that a certain percentage of flows have $\delta_i = 0$, leading to a cache hit rate of 0, and the rest of the flows in the network will have a low cache hit rate if their δ_i is small. Accordingly, the real flow setup latency of FlowN and OpenVirteX can be much larger than that provided in Figure 6. However, the flow setup latency of DeNH provided in Figure 6 represents the real flow setup latency of DeNH over an SDN system, because all the mapping records managed by a DeNH entity can be pre-loaded to the mapping table cache, which leads to a cache hit rate of 100%.

Table 4. Flows in the trace that only send their first packets to the control plane (%)

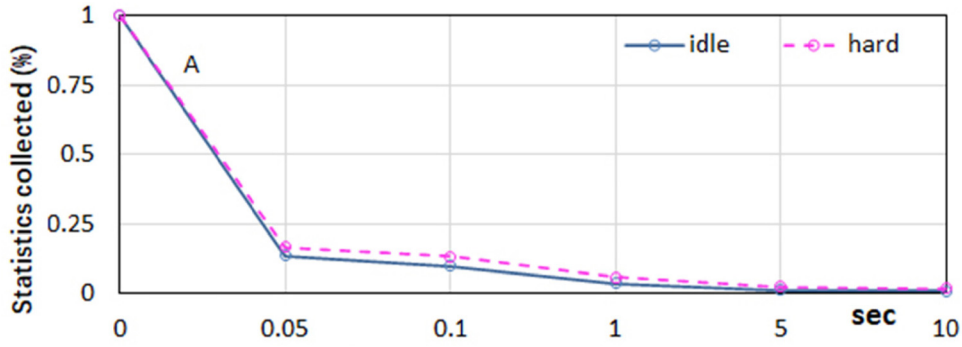
Timeout	0.05s	0.1s	1s	5s	10s
Idle	17.7	18.3	25.5	33.9	38
Hard	18.7	19.5	30.2	44.5	51.6

5.5 Flow Statistics

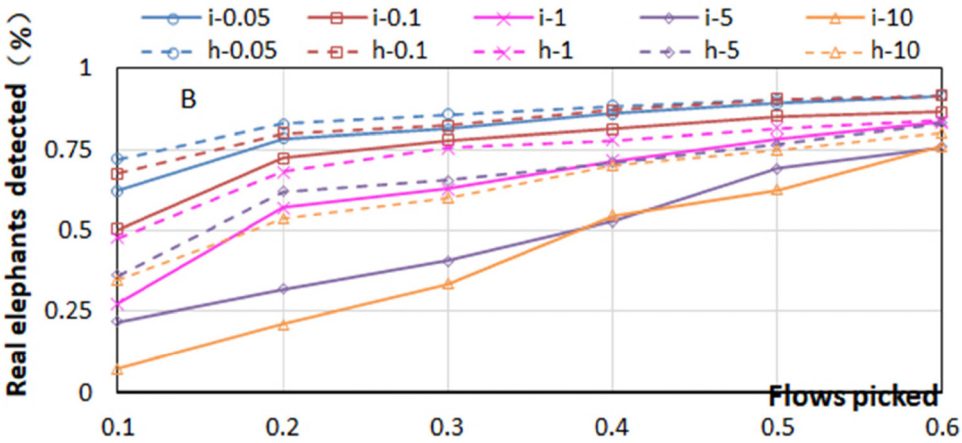
We firstly use the same packet trace as used in Section 5.4 and compute the percentage of the total number of packets of flows counted by the statistics using equation (5). As shown in Figure 7(a), the percentage is highly dependent on the timeout of flow entries. The shorter the timeout is, the more packets of flows are counted. This is because a shorter timeout makes more packets of flows sent to the control plane. Having the same timeout, the flow statistics under hard timeouts count more packets of flows than those under idle timeouts, because idle timeouts may make switches only forward the first packets of the elephant flows to the control plane, if the packets of the flows arrive frequently; whereas hard timeouts make switches forward the packets of the flows to the control plane periodically.

Although only less than 20% of the total packets of flows were sent to the control plane, we can use such flow statistics to detect the elephant flows in the network. Detecting the elephant flows in a network is crucial for optimizing the network resource usage, because elephant flows consume most of the network bandwidth. Particularly, we firstly sort the flows in the trace and in the collected flow statistics descendingly regarding the number of packets of a flow, respectively; then we select the top 10% to 60% of the flows from the trace as the base line, and compare them with the

top 10% to 60% of the flows selected from the collected flow statistics when the flow entries have various timeouts.



(a) Statistics collected by controllers



(b) Real elephants detected in the flow picked

Figure 7. Accuracy of the top flows picked by the flow statistics, i-idle timeout, h-hard timeout

As shown in Figure 7(b), when the timeout of flow entries was set to be 0.1s or less, a comparison between the sets of the top 30%, 40%, 50%, and 60% of the flows selected from the counted statistics and the trace was made, respectively. We found that 70%, 84%, 87%, and 90% of flows in two sets were matched, respectively. If we increased the timeout of flow entries to be 1s or larger, 50%, 60%, 70%, and 80% of the flows in the two sets were matched, respectively. The more candidates of elephant flows were selected from the statistics, the higher probability of the selected flows was the real elephant flows in the network. Such a probability is the accuracy of the statistics in detecting the elephant flows. In most of the time, a shorter timeout leads to a higher accuracy in the detection, and hard timeouts achieve higher detecting accuracy than idle timeouts. This is because a shorter idle timeout does not enforce an elephant flow with packets arriving frequently to send more packets to the control plane, while a shorter hard timeout does. Since most of the elephant flows can be identified by such detections, the flow statistics maintained by DeNH reflects the states of the network without adding large overhead on both control and data planes.

However, shorter timeouts enforce the control plane to be more frequently involved in forwarding lows, leading to an increased flow forwarding delay at switches. An advanced timeout adjustment mechanism should be developed in our future work to increase the accuracy of elephant flow detection without degrading the flow forwarding delay of switches [29]. Counting statistics may cause each mapping record to be updated every time it gets visited, leading to a huge cache syncronization overhead in DeNH [30]. In practice, DeNH should implement an individual module to count the statistics without involving the mapping table cache.

6 Conclusion

A distributed and efficient network hypervisor solution called DeNH has been proposed. It incorporates multiple entities to fully virtualize a network by converting network addresses and control logic between virtual and physical networks. DeNH provides resource abstraction and isolation based on logical ports of switches at the data plane, and supports a container-level isolation at the control plane. DeNH

maintains fully duplicates of global network topology and mapping table at each node to allow VNs to be embedded globally for better scalability. DeNH uses default tenant controller module to manage the VNs without extra requirements and concerns about control and security to increase the number of VNs an NH entity can support. DeNH provides a novel lightweight mapping table caching system with a record pre-fetching mechanism to increases the cache hit rate to 100% and effectively reduces the extra delay incurred by the conversion and the flow setup latency of the control plane. DeNH can maintain the statistics of the received packets of flows in its mapping table cache. The evaluations over emulated SDN systems have demonstrated the performance of DeNH and the flow statistics can give an accurate detection of elephant flows in the network.

Acknowledgements

This work was supported by the National Nature Science Foundation of China under Grant number 61962016.

References

[1] Open Networking Foundation, *Software-defined networking: The new norm for networks*, White Paper, pp. 2-6, April, 2012.

[2] I. Alam, K. Sharif, F. Li, Z. Latif, M. M. Karim, S. Biswas, B. Nour, Y. Wang, A Survey of Network Virtualization Techniques for Internet of Things Using SDN and NFV, *ACM Computing Surveys*, Vol. 53, No. 2, pp. 1-40, June, 2020.

[3] A. Blenk, A. Basta, M. Reisslein, V. Kellerer, Survey on network virtualization hypervisors for software defined networking, *IEEE Communications Surveys & Tutorials*, Vol. 18, No. 1, pp. 655-685, First Quarter, 2016.

[4] J. Li, D. Li, Y. Yu, Y. Huang, J. Zhu, J. Geng, Towards full virtualization of SDN infrastructure, *Computer Networks*, Vol. 143, pp. 1-14, October, 2018.

[5] D. Drutskoy, E. Keller, J. Rexford, Scalable network virtualization in software-defined networks, *IEEE Internet Computing*, Vol. 17, No. 2, pp. 20-27, March/April, 2013.

[6] B. Heller, *Openflow switch specification, version 1.0. 0. Wire*, ONF TS-001, December, 2009.

[7] A. Blenk, A. Basta, W. Kellerer, S. Schmid, On the impact of the network hypervisor on virtual network performance, *IFIP Networking*, Warsaw, Poland, 2019, pp. 1-9.

[8] A. Al-Shabibi, M. De Leenheer, M. Gerola, A. Koshibe, G. Parulkar, E. Salvadori, B. Snow, OpenVirteX: Make your virtual SDNs programmable, *the third workshop on Hot topics in software defined networking- HotSDN'2014*, Chicago, IL, USA, 2014, pp. 25-30.

[9] Y. Han, T. Vachuska, A. Al- Shabibi, J. Li, H. Huang, W. Snow, 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, e2012, March/April, 2018.

[10] L. X. Liao, A. Shami, V. C. Leung, Distributed FlowVisor: a distributed FlowVisor platform for quality of service aware cloud network virtualisation, *IET Networks*, Vol. 4, No. 5, pp. 270-277, September, 2015.

[11] E. Salvadori, R. D. Corin, A. Broglio, M. Gerola, Generalizing virtual network topologies in OpenFlow-based networks, *2011 IEEE Global Telecommunications Conference-GLOBECOM'2011*, Houston, TX, USA, 2011, pp. 1-6.

[12] R. D. Corin, M. Gerola, R. Riggio, F. De Pellegrini, E. Salvadori, Vertigo: Network virtualization and beyond, *2012 European Workshop on Software Defined Networking-EWSDN'2012*, Darmstadt, Germany, 2012, pp. 24-29.

[13] T. Koponen, K. Amidon, P. Balland, M. Casado, A. Chanda, B. Fulton, Network virtualization in multi-tenant datacenters, *the 11th USENIX Conference on Networked Systems Design and Implementation- NSDI'14*, Seattle, WA, USA, 2014, pp. 203-216.

[14] Z. Bozakov, P. Papadimitriou, Autoslice: automated and scalable slicing for software-defined networks, *the 2012 ACM conference on CoNEXT student workshop -ACM CoNEXT 2012*, Nice, France, 2012, pp. 3-4.

[15] L. X. Liao, C. F. Lai, J. Wan, V. C. M. Leung, T.-C. Huang, Scalable distributed control plane for On-line social networks support cognitive neural computing in software defined networks, *Future Generation Computer Systems*, Vol. 93, pp. 993-1001, April, 2019.

[16] L. X. Liao, H. C. Chao, M. Y. Chen, Intelligently modeling, detecting, and scheduling elephant flows in software defined energy cloud: A survey, *Journal of Parallel and Distributed Computing*, Vol. 146, pp. 64-78, December, 2020.

[17] S. Min, S. Kim, J. Lee, B. Kim, W. Hong, J. Kong, Implementation of an OpenFlow network virtualization for multi-controller environment, *14th International Conference on Advanced Communication Technology- ICACT 2012*, PyeongChang, Korea, 2012, pp. 589-592.

[18] P. Congdon, *Link layer discovery protocol and MIB. V1*, pp. 1-25, May, 2002.

[19] H. Keen, IEEE 802.1Q Virtual Bridged Local Area Networks, *IEEE Network,* Vol. 14, No. 4, pp. 3-3, July/August, 2000.

[20] P. Garg, Y. Wang, *NVGRE: Network virtualization using generic routing encapsulation*, RFC 7637, September, 2015.

[21] R. Sherwood, G. Gibb, K. K. Yap, G. Appenzeller, M. Casado, N. McKeown, G. Parulkar, *Flowvisor: A network virtualization layer*, OpenFlow Switch Consortium, Tech. Rep, OPENFLOW-TR-2009-1, October, 2009.

[22] T. Koponen, M. Casado, N. Gude, J. Stribling, L. Poutievski, M. Zhu, R. Ramanathan, Y. Iwata, H. Inoue, T. Hama, S. Shenker, Onix: A distributed control platform for large-scale production networks, *the 9th USENIX conference on Operating systems design and implementation- OSDI 2010*, Vancouver, Canada, 2010, Vol. 10, pp. 351-364.

[23] P. Berde, M. Gerola, J. Hart, Y. Higuchi, M. Kobayashi, T. Koide, B. Lantz, B. O'Connor, P. Radoslavov, W. Snow, G.

Parulkar, ONOS: towards an open, distributed SDN OS, *the third workshop on Hot topics in software defined networking-HotSDN 2014*, Chicago, IL, USA, 2014, pp. 1-6.

[24] T. Benson, A. Akella, D. A. Maltz, Network traffic characteristics of data centers in the wild, the *10th ACM SIGCOMM conference on Internet measurement- IMC'10*, Melbourne, Australia, 2010, pp. 267-280.

[25] S. C. Dake, C. Caulfield, A. Beekhof, The corosync cluster engine, *Linux Symposium*, Vol. 1, Ottawa, Ontario, Canada, 2008, pp. 85-99.

[26] Nicira Networks, *NOX network control platform*, https://github.com/noxrepo/ nox, retrieved in August 2019.

[27] C. Rotsos, N. Sarrar, S. Uhlig, R. Sherwood, A. W. Moore, OFLOPS: An open framework for OpenFlow switch evaluation, *In International Conference on Passive and Active Network Measurement*, Vienna, Austria, 2012, pp. 85-95.

[28] Data Set, 2010, http://pages.cs.wisc.edu/~tbenson /IMC10_Data.html, retrieved in August 2019.

[29] B. Isyaku, K. A. Bakar, M. S. M. Zahid, M. N. Yusuf, Adaptive and Hybrid Idle–Hard Timeout Allocation and Flow Eviction Mechanism Considering Traffic Characteristics, *Electronics*, Vol. 9, No. 11, pp. 1983, November, 2020.

[30] K. Gao, T. Nojima, H. Yu, Y. R. Yang, Trident: Toward Distributed Reactive SDN Programming With Consistent Updates, *IEEE Journal on Selected Areas in Communications*, Vol. 38, No. 7, pp. 1322-1334, July, 2020.

Biographies

Ling Xia Liao is currently a Full Professor with the School of Electronic Information and Automation, Guilin University of Aerospace Technology, China. She received her bachelor degree from Tsing Huang University, China and studied her MS and Ph.D degrees in the University of British Columbia, Canada. She has authored or co-authored over 20 refereed papers in journals, conferences, and workshop proceedings. She is the editor of the journal of Industry and Mine Automationt (Chinese core journal of science and technology). She is currently the PI and has participated many projects supported by National Nature Science Foundation of China, Natural Sciences and Engineering Research Council of Canada, and MITACs Canada. Her research interests include computer and network system technology, intelligent control technology, and big data analysis.

Jian Wang is currently an Full Associate Professor with the School of Information and Communication, Guilin University of Electronic Technology. She received her Bachelor and Master degrees from th e University of Electronic Technology, China. Her research interests include intelligent signal processing

Han-Chieh Chao is a Full Professor and Chair of the Department of Electrical Engineering, National Dong Hwa University, Taiwan, R.O.C. His research interests include High Speed Networks, Wireless Networks, IPv6 based Networks and Digital Divide. He received his MS and Ph.D. degrees from Purdue University. He has authored or co-authored 3 books and has published about 100 refereed professional research papers. Dr. Chao has received many research awards, including Purdue University SRC awards, and NSC research awards (National Science Council of Taiwan). He also received many funded research grants from NSC, Ministry of Education (MOE), RDEC, Industrial Technology of Research Institute, Institute of Information Industry and FarEasTone Telecommunications Lab.

Bin Qin is currently a Senior Engineer of the Department of Information Center, Guilin University of Aerospace Technology, China. He was graduated from South China University of Technology, China. His research interests include network management and optimization, information security, and big data analysis.