sustainability

Article

Cloud-Centric and Logically Isolated Virtual Network Environment Based on Software-Defined Wide Area Network

Dongkyun Kim [1], Yong-Hwan Kim [1], Ki-Hyun Kim [1] and Joon-Min Gil [2,*]

[1] Advanced Research Networking Center, Korea Institute of Science and Technology Information, Daejeon 34141, Korea; mirr@kisti.re.kr (D.K.); yh.kim086@kisti.re.kr (Y.-H.K.); kkh1258@kisti.re.kr (K.-H.-K.)
[2] School of Information Technology Engineering, Daegu Catholic University, Gyeongsan 38430, Korea
* Correspondence: jmgil@cu.ac.kr; Tel.: +82-10-3319-9071

Received: 15 November 2017; Accepted: 15 December 2017; Published: 20 December 2017

Abstract: Recent development of distributed cloud environments requires advanced network infrastructure in order to facilitate network automation, virtualization, high performance data transfer, and secured access of end-to-end resources across regional boundaries. In order to meet these innovative cloud networking requirements, software-defined wide area network (SD-WAN) is primarily demanded to converge distributed cloud resources (e.g., virtual machines (VMs)) in a programmable and intelligent manner over distant networks. Therefore, this paper proposes a logically isolated networking scheme designed to integrate distributed cloud resources to dynamic and on-demand virtual networking over SD-WAN. The performance evaluation and experimental results of the proposed scheme indicate that virtual network convergence time is minimized in two different network models such as: (1) an operating OpenFlow-oriented SD-WAN infrastructure (KREONET-S) which is deployed on the advanced national research network in Korea, and (2) Mininet-based experimental and emulated networks.

Keywords: SDN; SD-WAN; virtualization; cloud

1. Introduction

Auto-provisioning of dynamic and on-demand networking is basically required for end-to-end resource orchestration over a programmable and intelligent wide area network (WAN) in distributed cloud environments [1,2]. Therefore, software-defined wide area network (SD-WAN)—a specific application of SDN technology applied to wide area network (WAN) connections [3] for end-user sites, enterprises, campuses, and data centers via distributed central offices and/or regional centers of WAN service providers—becomes crucial. Compared to the conventional WAN infrastructure, the software-based control and distributed nature of SD-WAN can make it easier for networking resources to be virtualized and integrated into distributed cloud environments connecting geographically dispersed and multiple datacenters.

One of the most compelling aspects of distributed cloud environments based on wide area networks is reducing costs while offering redundancy, reliability, and geo-replication. Furthermore, other motivations of distributed cloud include reducing wide-area traffic and latency, efficient computation at the edge, and virtualization [4–6], while research challenges for networking are summarized as follows: distributed cloud communications in WAN, network virtualization, network security and privacy, high performance, edge network architecture with middleware, and scalability. In particular, secure access control in cloud computing is specified as a major issue to be figured out in terms of security and privacy based on cloud environment [7–9].

With regard to the abovementioned research challenges, this paper proposes a logically isolated and virtually converged network environment based on the virtually dedicated network (VDN) technology over sustainable SD-WAN. The proposed scheme is designed to offer a secured isolation of virtual networks with automated resource convergence, where resources include virtual machines (VMs), datacenters, and networks. In this way, VDN systems can provide dynamic and on-demand virtual network provisioning per user site, application, and service with specific network performance based on the transparent end-to-end path allocations and strict bandwidth guarantees. Furthermore, the proposed scheme incorporates two specific algorithms to manipulate virtual networks, i.e., to attain rapid generation and seamless reconfiguration of virtual networks.

To investigate and evaluate the scheme, we used both emulated networks and the actually operating (de facto) SD-WAN infrastructure. The latter is based on the representative national research network in Korea, named Korea Research Environment Open Network (KREONET) [10], which launched a sustainable SD-WAN initiative, KREONET Softwarization (KREONET-S) [11,12], in 2015. KREONET-S is not only a de facto sustainable public SD-WAN infrastructure in Korea, but also one of the singular international wide area software-defined networks in the world [13].

The rest of this paper is organized as follows. We start by explaining the status of the domestic and international KREONET-S deployment on KREONET in Section 2. Section 3 introduces the VDN generation operations and analysis based on two improved VDN algorithms for generating the isolated and secured data delivery tree. In Section 4, the VDN reconfiguration method is proposed. Section 5 describes the performance of the proposed algorithms, and Section 6 finally concludes this work.

2. Network Architecture and Core Components of KREONET-S

2.1. OpenFlow-Based Software-Defined Wide Area Network Deployment on KREONET-S

KREONET-S was established to implement a nationwide virtual programmable network infrastructure based on the open network operating system (ONOS) [14,15] and OpenFlow over a large-scale wide-area SDN [16]. It is designed to provide end-to-end SDN production network services for advanced research and applications, particularly for those who are requiring specified time-to-research and time-to-collaboration. KREONET-S is built on an ONOS-enabled distributed control platform, with specialized edge/access capabilities for international network operations and federation options.

So far, KREONET-S infrastructure has softwarized four regional and international network centers, which reside in three locations (Daejeon, Seoul, and Busan) in Korea, and one location (Chicago, IL) in the US, where the StarLight operates as an international and national communications exchange facility for advanced research and educations. Figure 1 shows the deployment status of KREONET-S. There are three primary components, namely OpenFlow-based SDN data plane networks, distributed SDN operating system (ONOS), and VDN application services.

Internationally, KREONET-S has made a long-distance wide area network connection to the StarLight facility in Chicago, IL in the US over a 100 Gbps optical fiber, which was mentioned as a first-of-a-kind international SD-WAN between Korea and the US. In Chicago, at the StarLight facility, KREONET-S is integrated to the international software-defined network exchange, enabling high-end research partners and organizations to optimally access resources in North America, South America, Asia, South Asia, Australia, New Zealand, Europe, and other sites around the world. In fact, the international SD-WAN over KREONET-S is almost 10,500 km apart between Korea and the US, where the round-trip time of packet delivery is approximately 155–165 ms. This new infrastructure has a great potential for verifying the multi-domain virtual network federations through ONOS and virtually converged network environment.

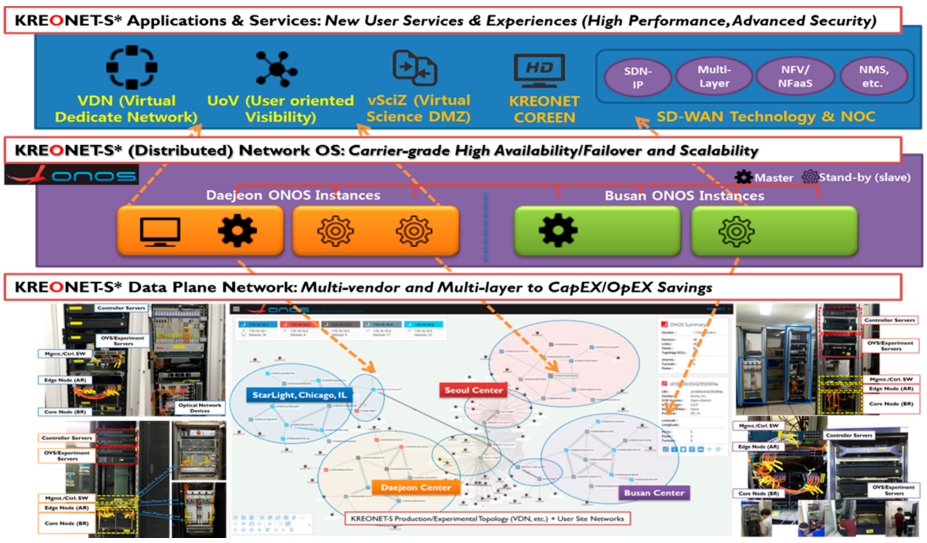

KREONET-S* Applications & Services: New User Services & Experiences (High Performance, Advanced Security)
VDN (Virtual Dedicate Network)
UoV (User oriented Visibility)
vSciZ (Virtual Science DMZ)
KREONET COREEN
SDN-IP
Multi-Layer
NFV/NFaaS
NMS, etc.
SD-WAN Technology & NOC
KREONET-S* (Distributed) Network OS: Carrier-grade High Availability/Failover and Scalability
onos
Master
Stand-by (slave)
Daejeon ONOS Instances
Busan ONOS Instances
KREONET-S* Data Plane Network: Multi-vendor and Multi-layer to CapEX/OpEX Savings
StarLight, Chicago, IL
Seoul Center
Daejeon Center
Busan Center
KREONET-S Production/Experimental Topology (VDN, etc.) + User Site Networks
Controller Servers
OVS/Experiment Servers
Mgmt./Ctrl. SW
Edge Node (AR)
Core Node (BR)
Optical Network Devices

Figure 1. KREONET Softwarization (KREONET-S) Deployment Status in Korea and the US, composed of three main building blocks (data plane, control plane, and application services).

2.2. *Virtually Converged Network Environment Based on VDN Functionality*

In this section, we describe the architecture and core functionalities of the virtually converged network environment based on VDN and user-oriented visibility (UoV) applications. The VDN/UoV applications are designed to create, update, and delete specific virtually isolated user group networks with deterministic QoS and data delivery performance guaranteed up to 100 Gbps over KREONET-S.

The VDN/UoV applications primarily enable the secured virtual network isolations, strict network bandwidth guarantees, and automated resource convergence. Figure 2 depicts the overall architecture of VDN/UoV applications where VDN conducts network virtualization and manipulations, and UoV provides user-oriented graphical interfaces and representational state transfer (REST) application programming interfaces (APIs) to converge various computing and storage resources in an intuitive, interoperable, and automated manner.

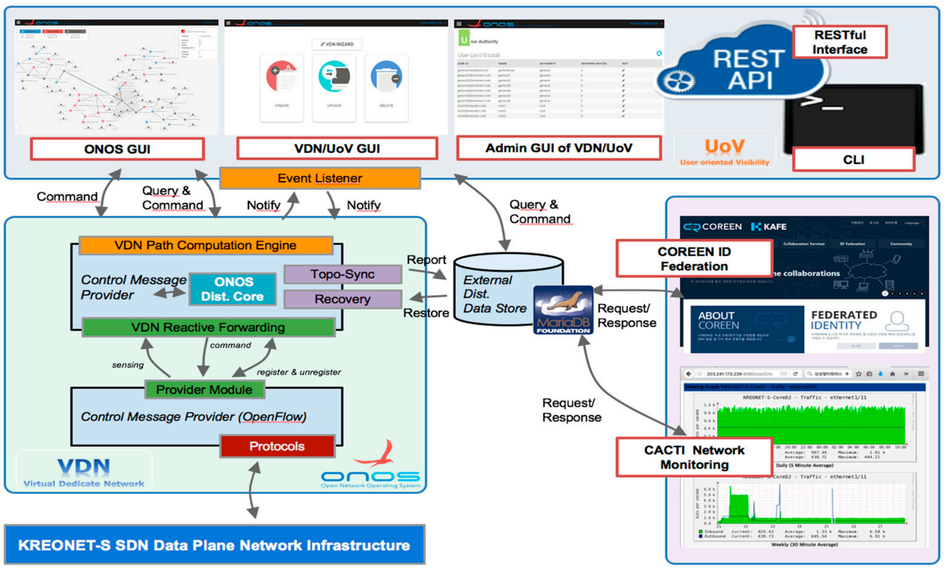


Figure 2. Virtually Converged Network Environment based on virtually dedicated network (VDN)/user-oriented visibility (UoV) Applications.

As shown in the figure, VDN/UoV applications consist of a path computation engine, a topology synchronization module, a link and topology recovery module, a VDN reactive forwarding engine, and user interfaces including graphical user interface (GUI), command-line interface (CLI), and REST APIs. Each component is aligned to manipulate (generate, update, and delete) the individual virtual networks by receiving user inputs (e.g., end hosts, required bandwidths, and authorized users), and detecting network events derived from ONOS, which controls OpenFlow-oriented data plane networks (e.g., link and topology up/down), etc. The UoV supports the corresponding user-friendly (and administrator's) GUI, CLI, and RESTful interfaces.

The abovementioned application modules are fundamentally related to rapidly generate the required dynamic VDNs on-demand, and to appropriately cope with the seamless reconfigurations of each virtual network per users' variable requirements without degrading the network performance. In this perspective, we focus on the fast VDN generation and seamless VDN reconfiguration methods in the following sections.

3. Scale-Aware Virtual Network Generation

In this section, we propose a method of dynamically generating VDNs on the SD-WAN such as KREONET-S as well as any emulated network. When an SD-WAN is deployed, we assume that SDN switches (also, denoted as SDN nodes) can be interconnected with multiple links, where each link has different network capacity (e.g., 1 Gbps to 100 Gbps). A VDN can be created on the SD-WAN with network resources (e.g., switches, ports, links, gateways to access mobile access networks [17]) allocated under the following conditions: (1) the end hosts joined in one VDN are constrained to communicate only with the other end hosts residing on the same VDN that provides logical network isolation and secure data delivery; (2) each end-to-end path between the end hosts in one VDN should strictly guarantee the network bandwidth required by end users (which offers deterministic network performance).

A VDN is generated as a data delivery tree (e.g., spanning tree) satisfying the above two conditions. The time complexity of constructing a data delivery tree is usually determined by network scale [18]. In this respect, to minimize the VDN generation time, first, we try to reduce the network scale by abstracting the SD-WAN with a pruning strategy and unification of multiple links. Next, we propose two improved VDN tree generation algorithms based on the highest closeness and node centrality between edge switches (directly connected to end hosts) inside each VDN.

3.1. Network Abstraction

As a first step of abstracting the SD-WAN or SDN network, we use a pruning strategy. Therefore, an SDN network $G(V, E)$ is abstracted to make a subgraph $G'(V', E')$ by pruning unnecessary end hosts, network nodes, and links, where V' is a set of network nodes with its available links and E' is a set of available links. The conditions for the available link are as follows:

1. A link is available if it can guarantee the required network bandwidth and it is not pre-allocated by other VDNs.
2. A link is available if its remaining network bandwidth is larger than the required bandwidth.

The 1st condition is required primarily to provide a guaranteed bandwidth provisioning service on the generated VDN. If pruning of $G'(V', E')$ by the 1st condition is not valid, reconstructing $G'(V', E')$ that meets the 2nd condition is performed. Here, the valid $G'(V', E')$ by the 1st condition, which is a subgraph of $G'(V', E')$ by the 2nd condition, means that all end-to-end paths between the requested VDN hosts can be found on the $G'(V', E')$. That is, all elements of $G'(V', E')$ by the 1st condition are included in $G'(V', E')$ by the 2nd condition. Therefore, in case of the 2nd condition, there are more candidate links for generating the VDN tree than the $G'(V', E')$ by the 1st condition. However, some links on $G'(V', E')$ by the 2nd condition can be already used in other VDNs. Therefore, the generated VDN based on $G'(V', E')$ by using the 2nd condition requires specific traffic engineering methods and/or traffic shaping techniques [19–21] for allocating link bandwidth as precisely as the demands of users. This causes additional overheads on the network system.

Figure 3a–c show an example of network abstraction with the network pruning method to meet the 1st condition in the case of the VDN generation requiring 10 G bandwidths with three end hosts. We first remove the end hosts and their links connected to the corresponding edge node. Then, the all the non-available links are removed at each of the edge node.

For SDN networks consisting of multiple links between their network nodes, only one link among multiple links between nodes should be allocated to some VDNs. Therefore, we choose one link with the minimum bandwidth between all node pairs in $G'(V', E')$, and then remove the other duplicated links before constructing a VDN data delivery tree. This is depicted in Figure 3d.

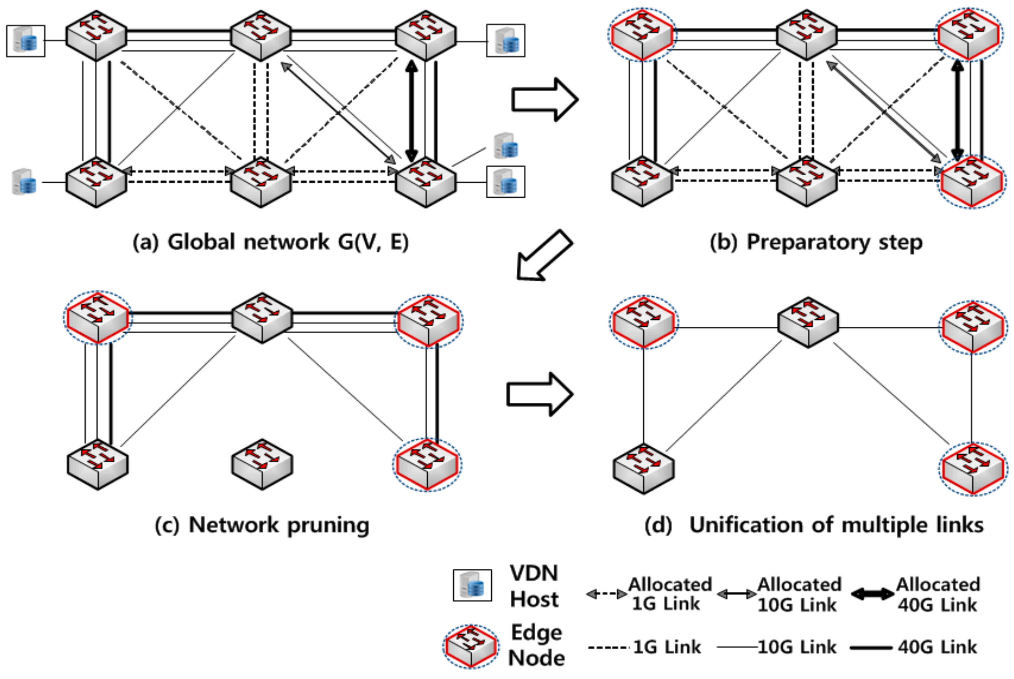


Figure 3. Network Abstraction in the Case of the VDN Required for 10 Gbps with three Hosts.

3.2. VDN Generation Algorithms Based on Node Centrality

In this subsection, we propose two VDN tree generation algorithms on the abstracted network $G'(V', E')$ by pruning the unnecessary network components corresponding to user requirements. Before presenting our solutions, we introduce several notations and variables that are mainly used in the rest of the paper. The notations are listed in Table 1.

Table 1. Annotation.

Notation	Meaning
$P_{i,j}$	Shortest path between node i and node j
n_c	center node with the highest closeness centrality
n_h	edge node of a host h
$N(H)$	set of edge nodes of a host set H
H_v	set of requested VDN hosts
H_p	set of previous VDN hosts
H_a	set of added VDN hosts
H_r	set of removed VDN hosts

In order to construct a VDN tree T in a general way, the shortest paths for all pairs of edge nodes related to the required VDN end hosts have to be calculated. For constructing T efficiently, we first select a center node n_c with the highest closeness centrality for edge nodes $N(H_v)$ of the required VDN end hosts H_v, and then calculate the shortest paths between the n_c and $N(H_v)$, and then merge the calculated paths into the VDN tree T. The detailed procedure is shown in Algorithm 1.

Algorithm 1. VDN Tree base on n_c in $G\prime(V', E')$

```
 0:   Parameter $G'(V', E'),\ N(H_v)$
 1:   Initialize $T$, $n_c$, sum, min = length of $G\prime(V', E')$;
 2:   **for each** $v \in V'$, and $v \notin N(H_v)$
 3:     **for each** $n \in N(H_v)$
 4:       Find $p_{v,n}$ on $G'$ and sum += the length of $p_{v,n}$;
 5:     **end for**
 6:     **if** ( sum < min)
 7:       min = sum and $n_c$ = $v$;
 8:     **end if**
 9:     sum = 0;
10:   **end for**
11:   **for each** $n \in N(H_v)$
12:    Find $p_{n,n_c}$ on $G'$ and $T = T \cup p_{n,n_c}$;
13:   **end for**
14:   **return** $T$
```

The node centrality approach not only provides a seamless VDN reconfiguration but also reduces the number of shortest path calculations for fast VDN generation. The former will be presented in next section.

Given a $G'(V', E')$ and H_v, the condition of selecting a center node n_c is as follows:

$$Min\left(\sum_{v \in V', \notin N(H_v)} \sum_{n \in N(H_v)} the\ lenght\ of\ p_{v,n}\right) \tag{1}$$

Most of the processing time of the VDN generation algorithm is required for calculating the shortest paths. In Algorithm 1, in order to generate a VDN, the number of shortest path calculations can be formulated as:

$$\left(|V'| - |N(H_v)|\right) \times |N(H_v)| + |N(H_v)| \times 1 \tag{2}$$

This depends on the size of the abstracted network $G'(V', E')$ and the size of the edge nodes $N(H_v)$. Therefore, the Algorithm 1 can result in poor performance if the size of $G'(V', E')$ becomes large. In order to improve this case, we propose Algorithm 2, which can select n_c in the subgraph consisting of plural shortest paths between $N(H_v)$. Thus, Algorithm 2 has good performance if the size of $N(H_v)$ becomes small regardless of the size of $G'(V', E')$. For Algorithm 2, the number of the shortest path calculations can be formulated as:

$$|N(H_v)| \times (|N(H_v)| - 1) + (|V''| - |N(H_v)|) \times |N(H_v)| + |N(H_v)| \times 1. \tag{3}$$

Algorithm 2. VDN Tree base on n_c in $G''(V'', E'')$

```
0:   Parameter $G'(V', E'),\ N(H_v)$
1:   Initialize $G''$, $T$, $n_c$, sum, min = length of $G\prime(V', E')$;
2:   **for each** $n1 \in N(H_v)$
3:     **for each** $n2 \in N(H_v)$
4:       Find $p_{n1,n2}$ on $G'$ and $G'' = G'' \cup p_{n1,n2}$;
5:     **end for**
6:   **end for**
7:   Perform Algorithm 1 ($G''(V'', E''), N(H_v)$)
```

Therefore, we can choose a more suitable algorithm according to the abstracted network size and requested VDN parameters. The condition for choosing Algorithm 2 can be expressed as:

$$\begin{aligned}
&(|V'| - |N(H_v)|) \times |N(H_v)| + |N(H_v)| \times 1 \ \geq \\
&|N(H_v)|(|N(H_v)| - 1) + (|V''| - |N(H_v)|) \times |N(H_v)| + |N(H_v)| \times 1 \\
&(|V'||N(H_v)|) - |N(H_v)|^2 \geq |N(H_v)|(|N(H_v)| - 1) + (\left|V''\right||N(H_v)| - |N(H_v)|^2 \\
&(|V'||N(H_v)|) \geq |N(H_v)|(|V''|+|N(H_v)| - 1) \\
&(|V'|) \geq |V''|+|N(H_v)| - 1.
\end{aligned} \tag{4}$$

In contrast to $|V'|$ and $N(H_v)$, $|V''|$ can be determined while performing Algorithm 2. Therefore, we should estimate $|V''|$ using the graph size based on the graph density or random walk techniques [22,23]. Those estimation methods can also be adjusted based on the network operation experience. However, the methods may estimate an approximate value of $|V''|$ with a margin of error. Fortunately, it is acceptable because the performance of the two algorithms is within the margin of error.

4. Selective VDN Reconfigurations

The VDN host group can be dynamically updated for new user demands. It means that the user may add and/or remove VDN hosts regardless of the host properties. For supporting new user demands properly, the VDN reconfiguration should meet the following requirements:

(1) The updated VDN hosts have to communicate with each other in the updated VDN;
(2) The end-to-end paths between updated VDN hosts should support the required bandwidths;
(3) The communications between previous VDN hosts except for the removed hosts should not be affected by the VDN reconfiguration;
(4) The process of VDN reconfiguration should be performed within the range of service tolerance.

In order to update the allocated bandwidths of the specific VDN, the network abstraction should be initialized for the VDN because the available link condition is changed. Therefore, we adopt a naive method that first releases the network resources (including flow rules) related to the current VDN, and then re-generate the VDN as indicated in Section 3. However, it causes the re-generated VDN topology to be in discord with the previous one. Therefore, the communications between the VDN hosts can be temporarily disconnected or delayed for a certain period of time (e.g., up to 1000 ms) when reestablishing the new communication paths on the updated VDN.

In this regard, we propose a selective VDN reconfiguration scheme considering the above four requirements based on pre-calculated VDN information. The proposed solution can be divided into three main steps: step (1) finding the center node with the highest closeness centrality in the VDN; step (2) identifying the hosts to be removed and releasing the network resources related to those hosts from the VDN; step (3) identifying the hosts to be removed and allocating the network resources related to those hosts to the VDN. Here, the first step can be skipped if the center node is already included in the VDN. Therefore, we will explain below the proposed scheme focusing on the remaining steps. The detailed procedure for the VDN reconfiguration is shown in Algorithm 3.

Step 2 of the selective VDN reconfiguration scheme works as follows (lines 1–10 in Algorithm 3). We should first find the removed VDN hosts H_r to release the network resources related to H_r from the VDN. It can be decided by comparing the set of requested VDN hosts H_n to the set of previous VDN hosts H_p. Given H_v and H_p, H_r is the difference set of H_v from H_p (H_r = $H_p - H_v$). For example, as shown in Figure 4, H_r = {$h1$, $h4$, $h7$} because of H_p = {$h1$, $h2$, $h3$, $h4$, $h7$}, and H_v = {$h2$, $h3$, $h5$, $h6$, $h8$}. If H_r is empty, we do not have to perform step 2 for the removed hosts.

Algorithm 3. Selective VDN Reconfiguration

```
0:  Parameter $G'(V', E')$, $T$, $n_c$, $H_p$, $H_v$
1:  $H_r = H_p - H_v$;
2:  if ($H_r$ is not empty)
3:    Initialize $T'$ and remove all flow rules related with $H_r$;
4:    for each $h \in H_p$
5:      if ($h \notin H_r$ and $< n_h \notin N(H_v)$)
6:        Find $p_{n_h,n_c}$ on $T$ and $T' = T' \cup p_{n_h,n_c}$;
7:      end if
8:    end for
9:    $T = T'$;
10: end for
11: $H_a = H_v - H_p$;
12: for each $h \in H_a$
13:   if ($n_h \notin T$)
14:     Find $p_{n_h,n_c}$ on $G'$ and $= T \cup p_{n_h,n_c}$;
15:   end if
16: end for
17: if ($T$ is changed)
18:   Reselect $n_c$ in $T$
19: end if
20: return $T$
```

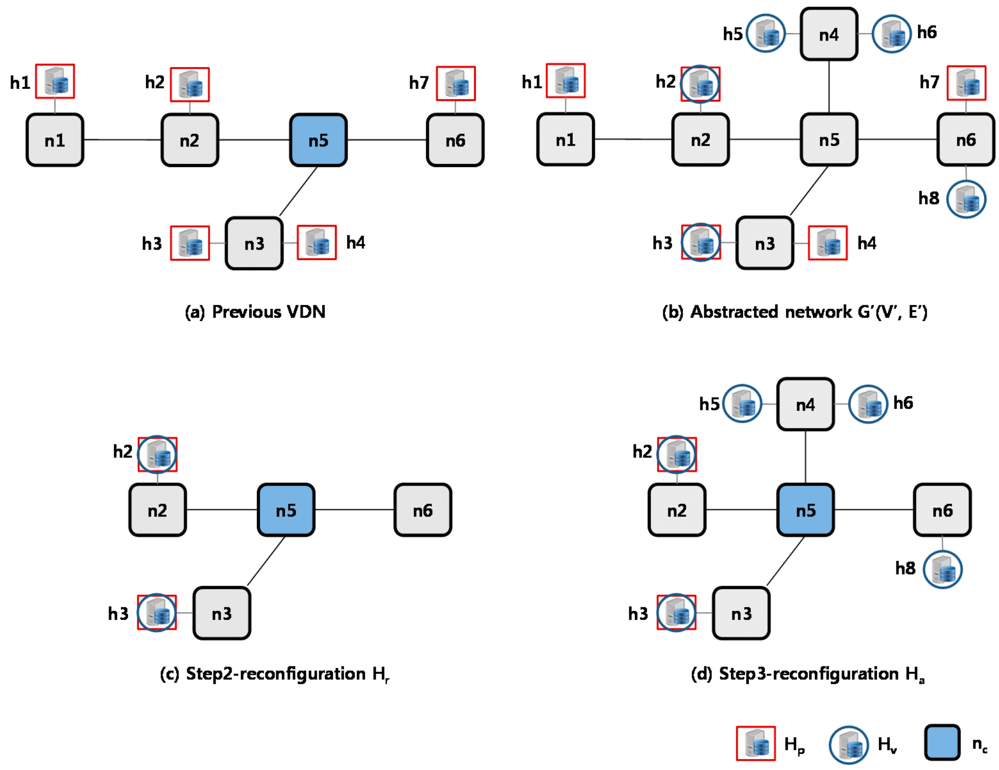

h1
h2
h7
n1
n2
n5
n6
h3
n3
h4
(a) Previous VDN
h5
n4
h6
h8
(b) Abstracted network G'(V', E')
(c) Step2-reconfiguration H_r
(d) Step3-reconfiguration H_a
H_p
H_v
n_c

Figure 4. Example of VDN reconfigurations. *Hp* = {*h1*, *h2*, *h3*, *h4*, *h7*}, Hv = {*h2*, *h3*, *h5*, *h6*, *h8*}, *Ha* = {*h5*, *h6*, *h8*}, and *Hr* = {*h1*, *h4*, *h7*}.

Once H_r is found, we eliminate all the flow rules related to H_r from the SDN nodes in the VDN. Moreover, the network resources for H_r should also be released from the VDN. However, this procedure will be unnecessary if an edge node n_h connecting to the removed host in H_r is included in $N(H_v)$

because the edge node will be used for other VDN hosts in H_v. Therefore, in this case, we only remove the flow rules related to the removed host. We can see those cases for *h4* and *h7* in Figure 4c. Otherwise, we should release the network resources related to the removed host in the VDN such as the edge node of *h1* in Figure 4c. For this, we calculate the shortest paths between n_c and $N(H_p)$ for all H_p except for the above hosts in the previous VDN (lines 4–8), and then merge the calculated paths into the updated VDN. This is because the nodes and links related to the remaining VDN hosts can be removed if the nodes and links are included in the paths between the center node and the edge nodes for the removed hosts such as the node of *h1* in Figure 4c.

In step 3, we handle the case of additional hosts H_a (lines 11–16 in Algorithm 3). For this, we should first find the set of additional VDN hosts H_a to allocate the network resources for H_a. It can also be decided by comparing H_v and H_p. Given H_v and H_p, H_a is the difference set of H_p from H_v ($H_a = H_v - H_p$) as shown in Figure 4, where $H_a = \{h5, h6, h8\}$. If H_a is empty, we do not have to perform step 3 for additional hosts similar to the procedure for H_r.

For the reconfiguration procedure of H_a, step 3 is simpler than that for H_r. It finds the new shortest paths on the abstracted network between the center node n_c and $N(H_a)$, and then merges it to the VDN. Furthermore, if an edge node n_h of additional hosts already belongs to T, we do not have to merge the path related to n_h. For this reason, the reconfiguration procedure of H_a should be performed next to that of H_r as shown in *h8* in Figure 4d. Finally, if T is changed, we reselect a new center node n_c based on the updated VDN using Equation (1).

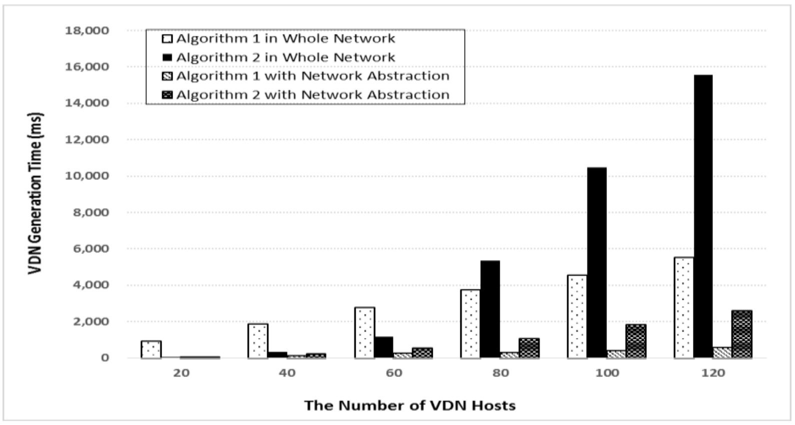
Concerning the processing time, the maximum number of shortest path calculations for each step can be expressed as $|N(H_p)| - |N(H_h)|$ and $|N(H_a)|$, respectively. In the real world, almost all the users reside in the same site (e.g., headquarter, branch, and main partner). Thus, the situation of adding and removing edge nodes is not frequent once VDNs are generated with the edge nodes included, although the characteristics of the nodes may change by the suggested pruning algorithm based on the two conditions in Section 3.1. That is, VDN re-configuration requires calculation of the new paths, otherwise they can be completed by just changing the set of VDN hosts.

5. Performance Evaluation and Experimental Results

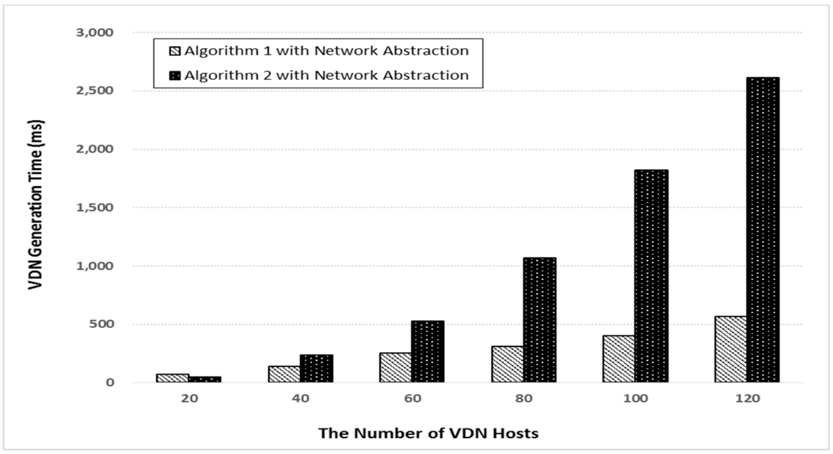
5.1. Evaluation Setup and Results Based on Mininet

For the performance evaluation of the proposed methods in the emulated networks, we set up an ONOS controller and Mininet emulator [24]. The ONOS controller manages and controls the virtual network topology generated by the Mininet emulator via the southbound interface and protocol (OpenFlow). The virtual network topology is created as a hybrid mesh and ring network composed of open virtual switches (OVSes). In addition, a virtual convergence network environment based on the VDN technology including the proposed methods is implemented in the form of an ONOS subsystem (application) as a northbound interface.

We first examine the VDN generation time in ms by increasing the number of VDN hosts as 20, 40, 60, 80, 100, and 120. Here, the entire network topology to be used in this evaluation consists of 50 nodes, 120 end-hosts, and 180 unidirectional links. The evaluation results in Figure 5 indicate that the VDN generation time of the proposed algorithms with network abstraction increases linearly as the number of VDN hosts participating in each VDN escalates on a scale of 20 to 120 by an interval of 20. On the other hand, the generation time without network abstraction is much higher than that in the applied cases. In Figure 5b, the VDN generation time when applying Algorithm 2 for network abstraction takes less than 2000 ms and 2700 ms when 100 and 120 number of hosts join a VDN, respectively, in the worst-case scenario. Note that it takes less than 600 ms when applying Algorithm 1.



(**a**) The cases of network abstraction/non-network abstraction

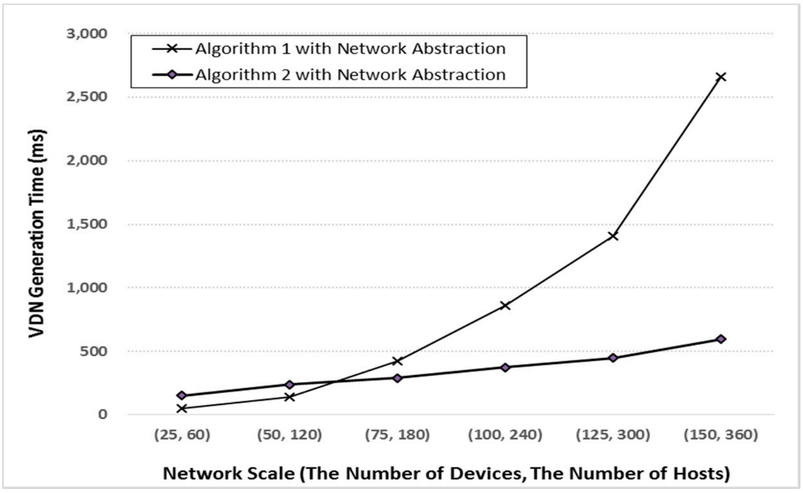


(**b**) The cases of specified network abstraction for proposed algorithms

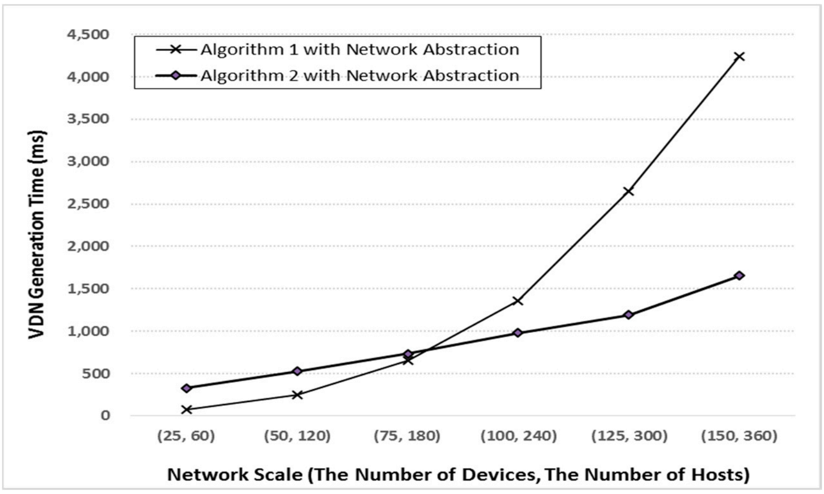
Figure 5. VDN generation time with the incremental number of VDN hosts.

Figure 6 depicts the VDN generation time with the variable network scale that is escalating on a scale of (25, 60) to (150, 360). Here, the number of participating VDN hosts is fixed in each VDN. In this simulation, the VDN generation time when applying Algorithm 1 increased exponentially according to the network scale. This is because the performance of Algorithm 1 is directly affected by the size of the network scale. On the other hand, the VDN generation time of Algorithm 2 with network abstraction increases slowly and linearly as the network scale varies because the performance of Algorithm 2 is influenced by $N(Hv)$. Thus, when the number of VDN hosts is increased, the performance gap decreases under the condition of a fixed network scale. Furthermore, as shown in Figure 6, the performance of Algorithm 1 is better than that of Algorithm 2 in the small-scale networks. As a result, we should select a suitable algorithm by using Equation (4) according to the network topology and required VDN parameters.

Figure 7 shows the VDN update time in the various cases with removed and added VDN hosts from 2 to 10 by an interval of 2. In Figure 7, the VDN regeneration means a naive method that simply releases the network resources related to the current VDN, and then regenerate the VDN as indicated in Section 3. In Figure 7a, the all edge nodes of H_r are included in $N(H_r)$. Accordingly, in the proposed algorithm, we just remove the flow rules of H_r and the H_r information from the set of VDN hosts. Therefore, the VDN update time for selective VDN reconfiguration is much smaller than the update time of the others. Figure 7b depicts the VDN update time when all edge nodes of H_r are not included in $N(H_r)$. Thus, in addition to releasing the network resources (flow rules and information for H_r), the new shortest paths between n_c and $N(H_r)$ have to be calculated. Even though an additional procedure is performed, the VDN update time is still much smaller than the update time of the others. The main reason for the performance difference between the proposed solution and the others is that the others are performed based on the entire network topology in contrast to the proposed solution, which is based on the pre-calculated VDN information. Figure 7c,d show the VDN update time according to the number of additional VDN hosts. In Figure 7c, all edge nodes of Ha already belong to the VDN to be updated in contrast to Figure 7d. Therefore, in Figure 7d, the new shortest path calculations are required for $N(H_a)$. According to the new path calculations, the VDN update time for the proposed solution in Figure 7d increases as the number of VDN hosts grows, compared to the update time of Figure 7c. However, in Figure 7c,d, we can observe that the VDN update time is much smaller than the update time of the others by similar reasons as those for Figure 7a,b. The update time of the selective VDN reconfiguration in Figure 7a,b is larger than that in Figure 7c,d because the former naively releases the network resources related to the VDN hosts.

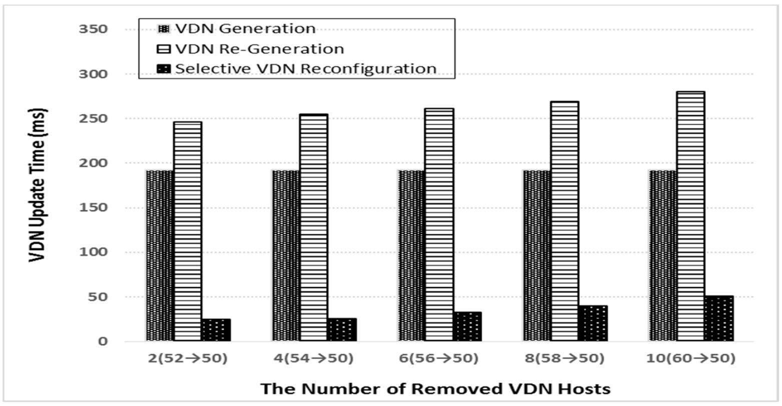


(**a**) The number of VDN hosts: 40

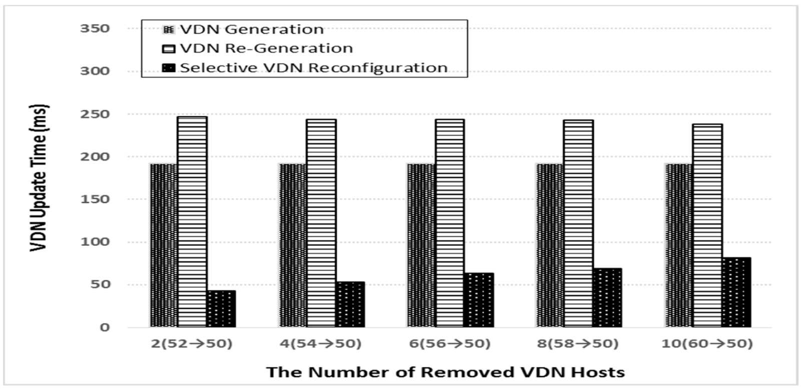


(**b**) The number of VDN hosts: 60

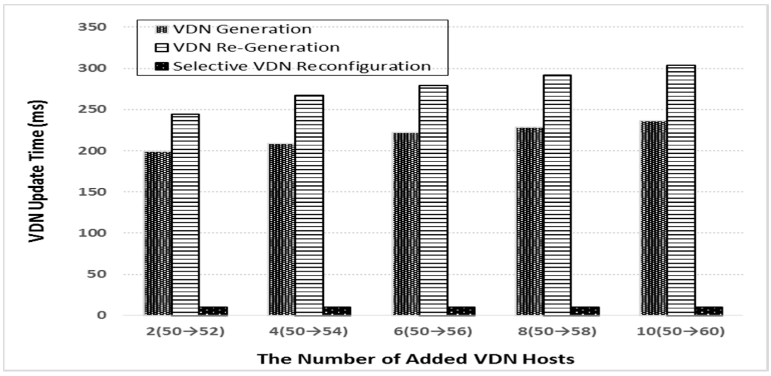
Figure 6. VDN generation time with the variable network scale.



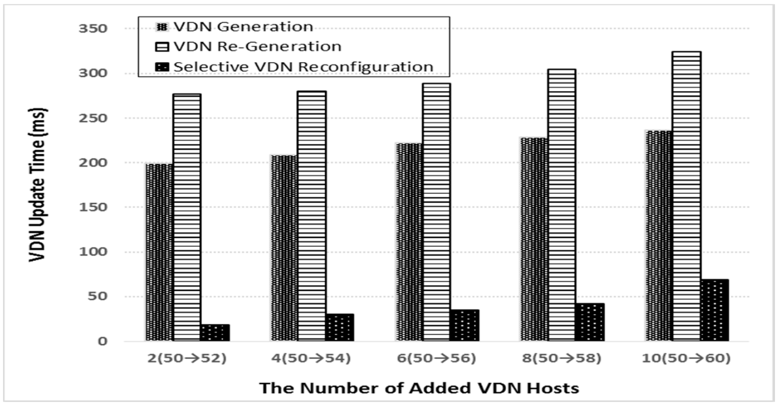
(**a**) All the edge nodes of H_r included in $N(H_r)$



(**b**) All the edge nodes of H_r not included in $N(H_r)$



(**c**) All the edge nodes of H_a already belonging to the VDN



(**d**) All the edge nodes of H_a already not belonging to the VDN

Figure 7. VDN reconfiguration time.

5.2. Evaluation Setup and Results Based on KREONET-S

KREONET-S is a de facto sustainable SD-WAN infrastructure deployed in four cities in Korea and the US. We used the operating SDN network infrastructure in the real world for the second evaluation setup and performance measurement. For the experimental setup, four core switches and four edge switches (in total, eight hardware network devices in Daejeon, Seoul, Busan, and Chicago) operating on KREONET-S are controlled through OpenFlow 1.3 protocol by sustainable and reliable five-instance ONOS cluster.

Furthermore, we have 40 additional OVSes attached to the corresponding physical core network devices and edge network devices, where the performance evaluation is conducted based on the five cases in Table 2. Each case has a set of parameters, namely variable number of network devices, end-hosts, and unidirectional links such as (8, 38, and 154) in case 1, (18, 40, and 218) in case 2, (28, 42, and 244) in case 3, (38, 44, and 308) in case 4, and (48, 45, and 368) in case 5.

According to the evaluation results in Figure 8, even for the worst case applied on KREONET-S (i.e., Case 5), it takes approximately 600 ms to create a VDN with 30 VDN hosts (VMs) participating. Here, each VM is selectively added to both physical and virtual SDN switches. As the automatically provisioned VDN can guarantee dedicated network bandwidths as shown in Figure 9 where the VDNs are provisioned with 1 Gbps and 10 Gbps dedicated paths, the results show that VDN users have potential to utilize the KREONET-S virtual network environment to ensure dynamic cloud resource convergence over wide area networks, with strict network isolation offered for higher data communications security for their advanced applications and research.

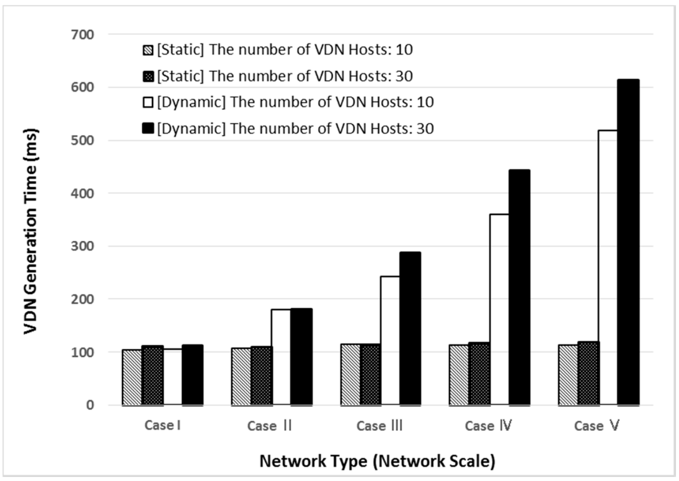
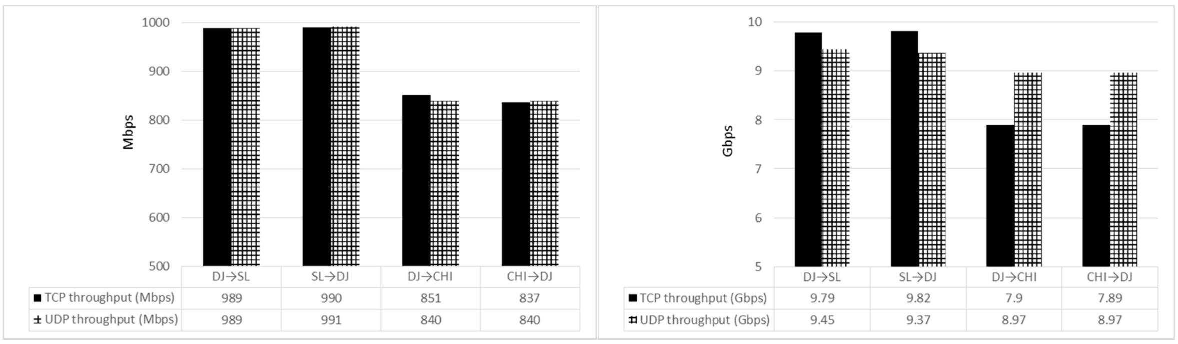


Figure 8. VDN generation time on KREONET-S.



	DJ→SL	SL→DJ	DJ→CHI	CHI→DJ
■ TCP throughput (Mbps)	989	990	851	837
▦ UDP throughput (Mbps)	989	991	840	840

	DJ→SL	SL→DJ	DJ→CHI	CHI→DJ
■ TCP throughput (Gbps)	9.79	9.82	7.9	7.89
▦ UDP throughput (Gbps)	9.45	9.37	8.97	8.97

(**a**)TCP and UDP Throughput on 1Gbps VDN (**b**) TCP and UDP Throughput on 10Gbps VDN

Figure 9. TCP and UDP traffic measurement for the VDNs with 1 Gbps and 10 Gbps bandwidths provisioned between Daejeon (DJ) in Korea, Seoul (SL) in Korea, and Chicago (CHI), IL, USA.

Table 2. Scale of network topology based on KREONET-S for each experimental case.

	Case I	Case II	Case III	Case IV	Case V
# of Nodes	8	18	28	38	48
# of VDN hosts	38	40	42	44	45
# of Unidirectional Links	154	218	244	308	368

6. Conclusions and Future Work

This paper proposes a new scheme of manipulating logically isolated virtual networks mainly for distributed cloud environments based on software-driven wide area networking. The proposed scheme describes the SD-WAN infrastructure and virtual networking application, which can allow dynamic and automated cloud resource convergence to be guaranteed on the demand of end-users, research groups and end-organizations.

Our performance evaluations conducted both in emulated networks and an actually operating (de facto) SD-WAN infrastructure indicated that sustainable virtual networks could be generated and reconfigured for the convergence of distributed cloud resources (e.g., VMs) dynamically in a relatively short time even in the worst-case scenario. Our future work includes an extended development of the virtual convergence network functionality as well as an improvement in the flexible and efficient federation with principal cloud computing software platform over SD-WANs. In addition, theoretical and practical comparison will be conducted to show the difference between the proposed pruning algorithm and classical algorithms such as Spanning Tree Protocol (STP), Kruskal, AHHK, etc.

Acknowledgments: This research was supported by Korea Institute of Science and Technology Information (KISTI).

Author Contributions: This research work was designed, carried out and written principally by Dongkyun Kim; Yong-Hwan Kim contributed mainly to the virtual network manipulations and algorithms; Ki-Hyun Kim contributed mostly to the experimental network testbeds for evaluations; Joon-Min Gil advised and commented for the overall architecture and design of the research work. All authors were involved in the finalization of the submitted manuscript. All authors read and approved the final manuscript.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Weldon, M.K. *The Future X Network: A Bell Labs Perspective*; Taylor & Francis Group, LLC.: Boca Raton, FL, USA, 2016; Chapter 5; pp. 160–193. ISBN 978-89-93712-73-5.
2. Manzalini, A.; Roberto, S.; Cagatay, B.; Prosper, C.; Slawomir, K.; Andreas, G.; Masake, F.; Eliezer, D.; David, S.; Mehmet, U. Software-Defined Networks for Future Networks and Services. In *White Paper Based on the IEEE Workshop SDN4FNS*; IEEE: Trento, Italy, 2014; pp. 5–14.
3. What is Software Defined WAN (or SD-WAN)? Available online: https://www.sdxcentral.com/sdn/definitions/software-defined-sdn-wan/ (accessed on 14 November 2017).
4. Coady, Y.; Hohlfeld, O.; Kempf, J.; McGeer, R.; Schmid, S. Distributed Cloud Computing: Applications, Status Quo, and Challenges. *ACM SIGCOMM Comput. Commun. Rev.* **2015**, *45*, 38–43. [CrossRef]
5. Jain, R.; Paul, S. Network Virtualization and Software Defined Networking for Cloud Computing: A Survey. *IEEE Commun. Mag.* **2013**, *51*, 24–31. [CrossRef]
6. Garcia Lopez, P.; Montresor, A.; Epema, D.; Datta, A.; Higashino, T.; Iamnitchi, A.; Barcellos, M.; Felber, P.; Riviere, E. Edge-centric Computing: Vision and Challenges. *SIGCOMM Comput. Commun. Rev.* **2015**, *45*, 37–42. [CrossRef]
7. Sun, L.; Ma, J.; Wang, H.; Zhang, Y. Cloud Service Description Model: An Extension of USDL for Cloud Services. *IEEE Trans. Serv. Comput.* **2015**, *99*, 1–14. [CrossRef]
8. Li, M.; Sun, X.; Wang, H.; Zhang, Y.; Zhang, J. Privacy-aware Access Control with trust management in Web Service. *World Wide Web* **2011**, *14*, 407–430. [CrossRef]
9. Wang, H.; Cao, J.; Zhang, Y. A Flexible Payment Scheme and its Role based Access Control. *IEEE Trans. Knowl. Data Eng. (TKDE)* **2005**, *17*, 425–436. [CrossRef]

10. KREONET Introductions. Available online: http://www.nisn.re.kr/eng/action.do?menuId=50030 (accessed on 8 December 2017).
11. Official KREONET-S Project Website. Available online: http://www.kreonet-s.net (accessed on 14 November 2017).
12. Kim, D.; Cho, H.; Kim, Y.; Kim, K.; Yu, K.; Gil, J. User-Oriented Software-Defined Wide Area Network Adopting Virtual Dedicate Networks. *ASP Adv. Sci. Lett.* **2016**, *22*, 2262–2267. [CrossRef]
13. KREONET-S Implements An SD-WAN Connection From South Korea to The StarLight International/National Communications Exchange Facility In Chicago: Inaugurating Novel Advanced International Communications Services. Available online: http://www.startap.net/starlight/PUBLICATIONS/KREONET-S.html (accessed on 14 November 2017).
14. Berde, P.; Gerola, M.; Hart, J.; Higuchi, Y.; Kobayashi, M.; Koide, T.; Lantz, B.; O'Connor, B.; Radoslavov, P.; Snow, W.; et al. ONOS: Towards an open, distributed SDN OS. In Proceedings of the Third Workshop on Hot Topics in Software Defined Networking (HotSDN 2013), Hong Kong, China, 2 January 2013; pp. 1–6.
15. Open Network Operating System. Available online: http://onosproject.org/ (accessed on 14 November 2017).
16. Nguyen, V.G.; Kim, Y.H. SDN-Based Enterprise and Campus Networks: A Case of VLAN Management. *J. Inf. Process. Syst.* **2016**, *12*, 511–524.
17. Gilani, S.M.M.; Hong, T.; Cai, Q.; Zhao, G. Mobility Scenarios into Future Wireless Access Network. *J. Inf. Process. Syst.* **2017**, *13*, 236–255.
18. Jung, H.; Kim, N. Multicast Tree Construction with User-Experienced Quality for Multimedia Mobile Networks. *J. Inf. Process. Syst.* **2017**, *13*, 546–558.
19. Sun, X.; Xie, G.G. An Integrated Systematic Approach to Designing Enterprise Access Control. *IEEE/ACM Trans. Netw.* **2016**, *24*, 3508–3522. [CrossRef]
20. Akyildiz, I.F.; Anjali, T.; Chen, L.; de Oliveira, J.C.; Scoglio, C.; Sciuto, A.; Smith, J.A.; Uhl, G. A new traffic engineering manager for Diff-Serv/MPLS networks: Design and implementation on an IP QoS testbed. *Comput. Commun.* **2003**, *26*, 388–403. [CrossRef]
21. OpenFlow® Switch Specification 1.3.1—Open Networking Foundation. Available online: https://3vf60mmveq1g8vzn48q2o71a-wpengine.netdna-ssl.com/wp-content/uploads/2013/04/openflow-spec-v1.3.1.pdf (accessed on 14 November 2017).
22. Gross, J.L.; Yellen, J. *Graph Theory and Its Applications*; CRC Press: Boca Raton, FL, USA, 1999.
23. Kurant, M.; Butts, C.T.; Markopoulou, A. Graph Size Estimation. *arXiv*. 2012. Available online: https://arxiv.org/abs/1210.0460 (accessed on 14 November 2017).
24. Mininet Website. Available online: http://mininet.org/ (accessed on 14 November 2017).