PAPER *Special Section on Network Virtualization, and Fusion Platform of Computing and Networking*

Dynamic Virtual Network Allocation for OpenFlow Based Cloud Resident Data Center

Tri TRINH[†a)], ***Student Member***, **Hiroshi ESAKI**[†b)], ***Member***, ***and*** **Chaodit ASWAKUL**[†c)], ***Nonmember***

SUMMARY Dynamic virtual network allocation is a promising traffic control model for cloud resident data center which offers virtual data centers for customers from the provider's substrate cloud. Unfortunately, dynamic virtual network allocation designed in the past was aimed to the Internet so it needs distributed control methods to scale with such a large network. The price for the scalability of the completely distributed control method at both virtual layer and substrate layer is the slow convergence of algorithm and the less stability of traffic. In this paper, we argue that the distributed controls in both virtual and substrate networks are not necessary for the cloud resident data center environment, because cloud resident data center uses centralized controller as the way to give network control features to customers. In fact, we can use the centralized algorithm in each virtual data center which is not very large network and the distributed algorithm is only needed in substrate network. Based on the specific properties of this model, we have used optimization theory to re-design the substrate algorithm for periodically re-adjusting virtual link capacity. Results from theoretical analysis, simulations, and experiments show that our algorithm has faster convergence time, simpler calculation and can make better use of the feedback information from virtual networks than the previous algorithm.

key words: *network virtualization, OpenFlow, data center, centralized controller, traffic management*

1. Introduction

The cloud resident data center [1], which has been proposed to replace the Amazon virtual private cloud (Amazon VPC) [2], offers customers not only a bunch of virtual machines but also the network services to connect and control traffics among these virtual machines. The restricted traffic control features provided by the Amazon VPC (e.g. statically add/remove routing entries, access control list) are not enough for the various requirements of customers. In fact, customers need the ability of freely choosing dynamic routing algorithms, security systems and caching systems to match with their operating model. A cloud resident data center offers those flexibilities by giving each of virtual data center customers a physical controller. This controller can then be used to enable those customers to install proper network control features to control their virtual network portion rented from the infrastructure cloud. There are three main properties of cloud resident data center to define the traffic management model. Firstly, a cloud resident data center is typically based on the physical network structure of distributed data centers each with a fat-tree topology. This allows the usage of cheaper switches in the network core layer than with a conventional topology requiring high-density-port switches for interconnections within each data center. This topology has many paths between each end-host pair so the multi-path traffic engineering is the most suitable for the traffic management model. Secondly, the traffic demands coming from virtual network customers are a priori unknown so static traffic controls can lead to higher chances of demand loss or capacity waste than dynamically adaptive traffic controls can. Thirdly, the requirement for virtual network provision to each customer suggests that cloud resident data centers should support the network virtualization [1]. The multi-path load balancing problem has been tackled in [3] by designing a traffic scheduler to estimate the elephant or big demands and using heuristic algorithms to search for the optimal mapping between these demands and physical paths. The dynamic congestion-control problem presented in [4] uses a marking system to decrease the source sending rate variation for archiving the high throughput even with a small buffer size. The combination of these two problems has been formulated and solved in the framework of centralized optimization algorithm [5]. However, all the aforementioned works have dealt with the traffic management problems in only one network layer. A cloud resident data center model needs the co-operation of traffic management mechanisms between virtual layer and substrate layer to maximize the total utility and make most uses of the physical resources. Davinci [6] has addressed this traffic management problem in two layers. At the virtual layer, an optimization problem has been formulated on the requirements of customers and solved distributedly by a decomposition method. At the substrate layer, a periodically updated mechanism has been proposed to control the capacity of virtual links according to the derived congestion prices. Unfortunately, this approach has been aimed at such big networks as the Internet so the distributed algorithms have been used in both substrate layer and virtual layer. This is not suitable for the traffic management problem in the cloud resident data center with virtualized networks each managed by a centralized controller dedicated for each of the virtual data center customers. This paper contains two original contributions.

Firstly, in this paper, the cloud resident data center testbed has been implemented in the OpenFlow platform. We have decided to use the OpenFlow because of its centralized system structure whose controller can have a global

Manuscript received May 18, 2012.
Manuscript revised August 30, 2012.
[†]The authors are with Chulalongkorn University, Thailand.
a) E-mail: tri.m@student.chula.ac.th
b) E-mail: hiroshi@wide.ad.jp
c) E-mail: chaodit.a@chula.ac.th
DOI: 10.1587/transcom.E96.B.56

view on the overall network. That centralized controller can give a better decision to manage the traffics inside the network. Additionally, the OpenFlow is the open system allowing devices from different vendors to in-operate smoothly via the OpenFlow standard. Our testbed has been implemented on Linux-based computers installed with OpenFlow switch [7] as the physical layer, FlowVisor [8] as the network virtualization layer, and Network Operating System NOX [9] as the centralized controller. All the system modules communicate via the OpenFlow [7] protocol to emulate the dynamics of a small cloud resident data center testbed.

Secondly, this paper re-designs the traffic control model in [6] for the OpenFlow-based network setting of cloud resident data center. Due to the centralized control structure of the OpenFlow, we have decided to implement the centralized optimization algorithm on each virtual network instead of distributed algorithm. And we have implemented the distributed traffic control algorithm in the substrate network to periodically control the allocation of virtual link capacity for the new model of cloud resident data center.

The rest of this paper is organized as follows. Section 2 describes the system model. Section 3 presents the derivation of our virtual link capacity updated algorithm for cloud resident data center. Section 4 demonstrates how we implement our algorithm in OpenFlow based Testbed. Section 5 shows the results and discussions. The final section is Conclusion and our future works.

2. System Description

Figure 1 shows the OpenFlow-based cloud resident data center system testbed constructed in this research. The system is comprised of four main elements, i.e., virtual network controller, substrate node, end-host and host manager. The function of virtual network controller is to control how substrate nodes switch the packet flows to suitable destinations. The substrate nodes switch the packet flows in accordance with the instructions from the virtual network controller and periodically calculate as well as update to the

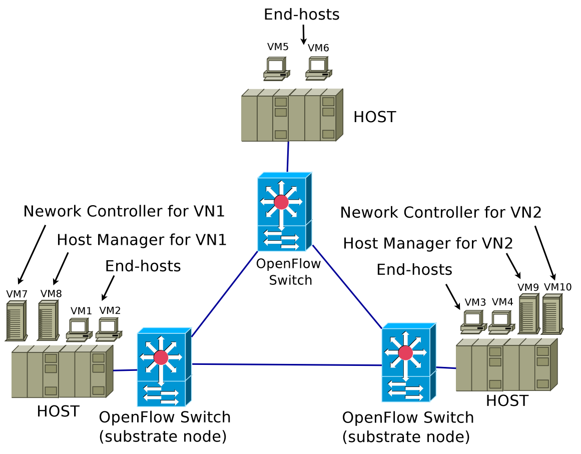


Fig. 1 System functional description.

controller about the new capacity required for their virtual links. An end-host is a virtual machine on a virtual network. Each end-host gets traffic optimal rates and multi-path splitting ratio from its host manager to adjust its outgoing traffic rates. The host manager calculates and controls the traffic rate of all end-hosts in its virtual network in response to the demands requested from end-hosts and the virtual network state updated by the controller.

2.1 Traffic Control in Each Virtual Network

Initially, the virtual network controller creates tunnels to connect its virtual network's end-hosts by adding the appropriate switching entries to the OpenFlow switches. Each end-host monitors its aggregated traffic data at its output port and sends the capacity request to its host manager periodically. On receiving the requests from all of its end-hosts, the host manager will calculate the optimal rate for each tunnel and reply these optimal rate back to the end-hosts. The end-hosts can then adapt the traffic sending rates to the tunnels accordingly. With this mechanism, the applications on each end-host does not need to wait for the optimal rate from host manager but they can send their traffic out immediately each time they want.

2.2 Traffic Control in Substrate Network

The mission of traffic control in a substrate network is to adjust the virtual link capacity periodically in order to maximize the total utility. In this paper, the substrate network uses congestion price of each virtual link to calculate the suitable bandwidth provided to virtual links. Here, a substrate node does not need to calculate the congestion price for virtual links one by one by itself as previously implemented in [6]. Instead, in this paper, the substrate node collects the congestion price for all virtual links from the host manager. The optimization solver in the host manager gives the solution of dual variables, which are congestion prices, while solving the central optimization problem for end-host traffic sending rate. This way of collecting congestion price information makes our algorithm outperform [6] because the substrate nodes do not need the computational resources to calculate the congestion price for each virtual link and they do not need to spend a lot of time to wait for the algorithm of calculating congestion price to converge. So, our algorithm can re-adjust the virtual link capacity faster and hence more frequently than the approaches in [6].

3. Proposed Optimization Framework for Cloud Resident Data Center

3.1 Modeling and Notations

The substrate network is modeled by a weighted, directed graph $G(V^s, E^s)$, where V^s is the set of all substrate nodes and E^s is the set of all directed substrate links. In Fig. 2, there are three substrate nodes indexed by $\{1, 2, 3\}$ and six

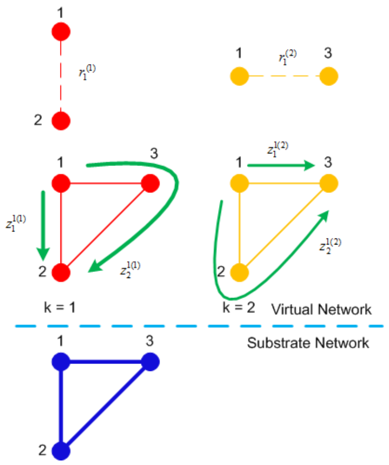


Fig. 2 System modeling example.

directed substrate links indexed by $\{1, 2, 3, 4, 5, 6\}$ between substrate node pairs $\{1-2, 2-1, 1-3, 3-1, 2-3, 3-2\}$, respectively. Similarly, a virtual network k is modeled as a graph $G(V^{(k)}, E^{(k)})$, where $k \in K$ is an element in the virtual network index set. Here, $V^{(k)}$ is the set of nodes in virtual network k and $E^{(k)}$ is the set of virtual links in virtual network k. For example, in Fig. 2, there are two virtual networks, the yellow one and the red one, each with the same topology as that of substrate network. Let $r_i^{(k)}$ denote the demand i of virtual network k. The virtual network controllers use their own traffic engineering algorithm to control the traffic flowing through their networks. And the substrate network controls the virtual link capacity by using virtual link congestion price information. For the example model in Fig. 2, there is one demand on each of the two virtual networks, denoted as $r_1^{(1)}$ and $r_1^{(2)}$, respectively. There are two paths $\{1-3, 1-2-3\}$ for demand $r_1^{(1)}$ and two paths $\{1-2, 1-3-2\}$ for demand $r_1^{(2)}$.

Define l as the general link index, where $l \in L$ denotes the substrate link in the substrate network and $l \in L_k$ denotes the virtual link in virtual network k. Link l of substrate network has the capacity of c_l. Link l of virtual network k has capacity $y_l^{(k)}$. For virtual network k, each demand can usually be assigned along more than one path at the same time so let us define $z_j^{i(k)}$ as the max flow rate available on path j for demand i of virtual network k. Usually, this flow rate must be less than or equal to the instantanenous amount of capacity on path j that is assigned to that demand i to prevent any possible traffic losses. The aim of our optimization is to find the proper values of these capacity and flow rate allocation parameters $y_l^{(k)}$ and $z_j^{i(k)}$.

To define the set of all possible paths whose capacity and flow rate allocation will be found by our proposed optimization, we use the m-shortest path first algorithm [10] to find m least cost paths from each source to each destination in the substrate network as well as in the virtual networks. Then we restrict our optimization search space within those obtainable paths only.

The link-path routing matrix for virtual network k [11] with its column representing each path in source-destination pairs and its row representing each substrate link index is denoted by $\mathbf{H}^{(k)}$, where $H_{lj}^{i(k)} = 1$ if demand i of virtual network k on path j uses virtual network link l and $H_{lj}^{i(k)} = 0$ otherwise. For example, in Fig. 2, there are two routing matrices $\mathbf{H}^{(1)}$ for virtual network 1 and $\mathbf{H}^{(2)}$ for virtual network 2. Here, $\mathbf{H}^{(1)} = [0\ 1; 0\ 0; 1\ 0; 0\ 0; 0\ 1; 0\ 0]$ because demand 1 of virtual network 1 on path 1 uses virtual network links 3 and demand 1 of virtual network 1 on path 2 uses virtual network links 1 and 5. Similarly, the second routing matrix $\mathbf{H}^{(2)} = [1\ 0; 0\ 0; 0\ 1; 0\ 0; 0\ 1; 0\ 0]$.

3.2 Virtual-Network Centralized Algorithm

For each fixed k with fixed values of $y_l^{(k)}$:

$$\begin{aligned}
\text{maximize:}\quad & U^{(k)}\left(z_j^{i(k)}, y_l^{(k)}\right)\\
\text{subject to:}\quad & \sum_i \sum_j H_{lj}^{i(k)} z_j^{i(k)} \le y_l^{(k)} \quad \forall l \in L_k\\
& z_j^{i(k)} \ge 0 \quad \forall i, j\\
\text{variables:}\quad & z_j^{i(k)}
\end{aligned} \tag{1}$$

As in [6], the objective function of the virtual network centralized problem (1) can be any concave utility function which varies with the capacity allocation parameters. The first constraint implies for each virtual network that the instantateous capacity requirement or total flow rate from all demands on their paths sharing a given virtual link cannot exceed the capacity assigned for that virtual link by the substrate network. The second constraint states that the capacity allocations must be non-negative.

3.3 Substrate-Network Distributed Algorithm

Our starting point is the same as [6] but with the setting of cloud resident data center in mind, we have derived a different virtual link capacity updated algorithm to match within our OpenFlow based network testbed. Our centralized substrate network optimization is stated by:

$$\begin{aligned}
\text{maximize:}\quad & \sum_k w^{(k)} U^{(k)}\left(z_j^{i(k)}, y_l^{(k)}\right)\\
\text{subject to:}\quad & \sum_i \sum_j H_{lj}^{i(k)} z_j^{i(k)} \le y_l^{(k)} \quad \forall l \in L_k, \forall k\\
& \sum_k y_l^{(k)} \le c_l \quad \forall l \in L\\
& y_l^{(k)}, z_j^{i(k)} \ge 0 \quad \forall i, j, k\\
\text{variables:}\quad & z_j^{i(k)}, y_l^{(k)}
\end{aligned} \tag{2}$$

Here, $w^{(k)}$ is the weight for each virtual network k. By using the centralized solver for each virtual network, we can derive the new virtual link capacity updated algorithm which is expectedly converging faster and better in processing bursty traffics than that of [6].

The problem in (2) is the convex problem because the objective function of this problem is the positive-weighted sum of concave objective functions (1) and the constraints are affine with $z_j^{i(k)}, y_l^{(k)}$. The problem has coupled variables $z_j^{i(k)}, y_l^{(k)}$ so we decouple it by the primal decomposition method [12]. We fix the the variable $y_l^{(k)}$ to make the problem only varies with $z_j^{i(k)}$ and the new problem can be divided into many centralized problems of (1). The optimization solver CVXOPT [15] which implements the so called primal-dual interior point method [13] is used to solve each problem of (1) centrally in the controller of each virtual network.

The results of optimization solver for each sub-problem are optimal flow rates $z_j^{*i(k)}$ and link congestion price $\lambda_l^{*(k)}$. Now, after solving the sub-problem, we need to find the new virtual link capacity $y_l^{(k)}$ that we have fixed in the previous step. The updated algorithm is derived from the master problem as follows:

$$\begin{aligned}
\text{maximize:} \quad & \sum_k w^{(k)} U^{(k)} \left(z_j^{*i(k)}, y_l^{(k)}\right) \\
& \sum_k y_l^{(k)} \leq c_l \quad \forall l \in L \\
& y_l^{(k)} \geq 0 \quad \forall k, l \\
\text{variables:} \quad & y_l^{(k)}
\end{aligned} \tag{3}$$

The above master optimization problem is also the convex problem because it has the same concave objective function as (2) and the affine constraints. As suggested in [13] for the convex optimization problem, because it is easy to project on the non-negative orthant constraint $y_l^{(k)} \geq 0$, we simplify the way to solve the problem (3) by considering the constraint $y_l^{(k)} \geq 0$ as implicit constraint and solve the problem with explicit constraint first then project the results on the implicit constraint domain. The problem with explicit constraint has coupled constraint $\sum_k y_l^{(k)} \leq c_l$ so we use the dual decomposition [12] to decouple it. We find the Lagrange function of (3) as follow:

$$\begin{aligned}
&\mathcal{L}\left(y_l^{(k)}, \mu_l\right) \\
&= \sum_k w^{(k)} U^{(k)} \left(z_j^{*i(k)}, y_l^{(k)}\right) - \sum_l \mu_l \left(\sum_k y_l^{(k)} - c_l\right) \\
&= \sum_k w^{(k)} U^{(k)} \left(z_j^{*i(k)}, y_l^{(k)}\right) - \sum_k \sum_l y_l^{(k)} \mu_l + \sum_l \mu_l c_l
\end{aligned} \tag{4}$$

Note that μ_l is the dual variable for the master problem which is different from $\lambda_l^{(k)}$, the dual variable of subproblems.

We use the subgradient projection method to find the supremum of problem (4) which has implicit constraint of non-negative orthant and the variable of $y_l^{(k)}$. The update function for $y_l^{(k)}$ is

$$y_l^{(k)}(t + T_s) = \left[y_l^{(k)}(t) + \alpha \frac{\partial}{\partial y_l^{(k)}} \mathcal{L}\left(y_l^{(k)}, \mu_l\right) \right]^+ \tag{5}$$

where T_s is the time period for updating virtual link capacity in the substrate network and the updated time argument has been appended in the parenthesis following the variable $y_l^{(k)}$. $[]^+$ denotes the projection on to non-negative orthant domain.

Like [13], we have the partial derivative of objective function at the optimality as the link congestion price so

$$\frac{\partial}{\partial y_l^{(k)}} \left(\sum_{k \in K} U^{(k)} \left(z_j^{*i(k)}, y_l^{(k)}\right) \right) = \lambda_l^{*(k)} \tag{6}$$

Using (6) we have partial derivative of Lagrange function as

$$\frac{\partial}{\partial y_l^{(k)}} \mathcal{L}\left(y_l^{(k)}, \mu_l\right) = w^{(k)} \lambda_l^{*(k)} - \mu_l \tag{7}$$

To find μ_l, we define $p^{*(k)}$ as the optimal value of (1) and p^* as the optimal value of (3). We have

$$\begin{aligned}
p^* &= \sum_k w^{(k)} p^{*(k)} \\
\frac{\partial p^*}{\partial c_l} &= \sum_{k \in K_l} w^{(k)} \frac{\partial p^{*(k)}}{\partial c_l} \\
\mu_l^* &= \sum_{k \in K_l} w^{(k)} \lambda_l^{*(k)} \frac{\partial y_l^{(k)}}{\partial c_l}
\end{aligned} \tag{8}$$

where K_l is the set of virtual networks hosted on substrate link l.

In practical networks, a service provider prefers to allocate all their capacity fairly to increase the quality of services. In our system, by constraining on $c_l = \sum_{k \in K_l} y_l^{(k)}$, we allocate fairly all of our residual capacity to virtual network customers:

$$\frac{\partial y_l^{(k)}}{\partial c_l} = \frac{y_l^{(k)}}{\sum_{\kappa \in K_l} y_l^{(\kappa)}} \tag{9}$$

Hence, we have from (8), (9) that

$$\mu_l^* = \frac{\sum_{k \in K_l} w^{(k)} \lambda_l^{*(k)} y_l^{(k)}}{\sum_{k \in K_l} y_l^{(k)}} \tag{10}$$

And the final virtual link capacity update function can be obtained by combining (5),(7),(10) :

$$\begin{aligned}
&y_l^{(k)}(t + T_s) \\
&= y_l^{(k)}(t) + \alpha \left[w^{(k)} \lambda_l^{*(k)} - \frac{\sum_{k \in K_l} w^{(k)} \lambda_l^{*(k)} y_l^{(k)}(t)}{\sum_{k \in K_l} y_l^{(k)}(t)} \right]^+
\end{aligned} \tag{11}$$

As a summary, in (11), we can find the value of $y_l^{(k)}$ at time $t + T_s$ from the previously updated value of $y_l^{(k)}$ at time t. In the recursion, α is a constant stepsize that can be chosen through experiments to improve the algorithm convergence. The value of link congestion price $\lambda_l^{*(k)}$ in virtual network k can be obtained by centralized optimization solver. This equation also has a reasonable interpretation because if the weighted rate of utility increase per unit of capacity being added for a given virtual link l, i.e. the term $w^{(k)}\lambda_l^{*(k)}$, is greater than the rate of utility increase as averaged over all the virtual link capacity upgrade, then that virtual link l of the virtual network k should be allocated more capacity. On the contrary, if the weighted rate of utility increase per unit of capacity being added for a given virtual link l is lower than the rate of utility increase as averaged over all the virtual link capacity upgrade, then that virtual link l should be assigned less capacity.

3.4 Convergence and Optimality

Theorem: The updated Eq. (11) converges to optimal point of (2) if:

- The problem (1) is convex optimization.
- The virtual link bandwidth allocation is updated after the convegence of (1)
- The stepsize α in (11) is small enough or diminishing (diminishing stepsize usually converge faster constant stepsize).

Proof: From the above interpretation of (11) about the principal for increasing and decreasing bandwidth of each virtual link $y_l^{(k)}$, the aggregated utility on each physical link l and hence, the total utility overall substrate network: $\sum_k w^{(k)}U^{(k)}(z_j^{*i(k)}, y_l^{(k)})$ will be increased after each virtual link capacity updated period T_s. Because we always keep $c_l = \sum_k y_l^{(k)}$, $\mathcal{L}(y_l^{(k)}, \mu_l)$ increases monotonically after every period of updating virtual link capacity. But, $\mathcal{L}(y_l^{(k)}, \mu_l)$ is concave function so it is only able to increase until the maximum point which also be the stationary point when $y_l^{(k)}$ moves to the limited point $\tilde{y}_l^{(k)}$. At the limited point we have:

$$w^{(k)}\lambda_l^{*(k)} = \frac{\sum_{k\in K_l} w^{(k)}\lambda_l^{*(k)}\tilde{y}_l^{(k)}}{\sum_{k\in K_l}\tilde{y}_l^{(k)}} \qquad (12)$$

It means that at the limited point $\tilde{y}_l^{(k)}$, the weighted rate of utility increase per unit of capacity being added for a given virtual link l is equal to the rate of utility increase as averaged over all the virtual link capacity upgrade and $y_l^{(k)}$ does not change anymore.

At the $\tilde{y}_l^{(k)}$:
$\frac{\partial}{\partial y_l^{(k)}}\mathcal{L}(\tilde{y}_l^{(k)}, \mu_l) = 0 \;\; \forall l, k;\;\; \sum_k y_l^{(k)} \le c_l$, and $\mu_l(\sum_k y_l^{(k)} - c_l) = 0$. The KKT conditions [13] are satisfied for the convex problem of (3), then the limited point of (11) is the optimal point of (3).

Because Eq. (1) is converged before the virtual link bandwidth update time, at $y_l^{(k)} = \tilde{y}_l^{(k)}$, we also have $z_j^{i(k)} = z_j^{*i(k)}$ and they are the optimal point $y_l^{*(k)}$ and $z_j^{*i(k)}$ for the problem of (2).

4. Implementation

4.1 General Testbed Description

We have built a testbed to implement and evaluate our optimization algorithm with a rack of seven computers named PC1-5 and TA, TB, respectively. Each computer is equipped with Core 2 quad 2.40 GHz and 4 GBytes of RAM. Ubuntu OS 11.04 with the core of 2.6.38 has been installed on each physical computer.

4.2 Logical Model

The logical model of our network is presented in Fig. 3. In this model, three substrate nodes have been created by installing OpenFlow switch software v0.8.9r2 on computers PC1-3 to make them work as PC-based switches. FlowVisor has been used to create a virtualization layer on those substrate nodes. The virtual network controller has been instantiated by a Python script running on top of the network operating system NOX. We have written a Python script to add the switching entries to OpenFlow switches and control them to forward packets according to VLAN IDs. To implement end-hosts to split the traffic through paths and limit the rate of outgoing traffics, we have written a Python script using Scapy [14] to create two kinds of packets with VLAN ID = 2 and VLAN ID = 3. And the end-hosts send the packets to output ports according to the control information from the central host manager. This host manager has been implemented on PC5 by a Python script which communicates with all end-hosts and the controller in the corresponding virtual network. This script periodically takes traffic demands from end-hosts and virtual link capacity value updates from substrate nodes as its inputs and use CVX-OPT [15] to calculate the optimal rates for all end-hosts

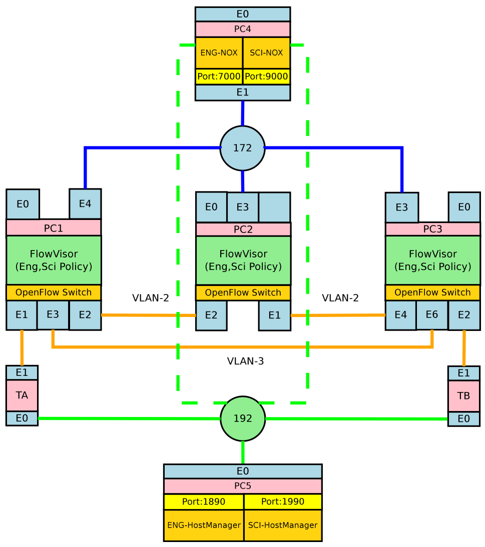


Fig. 3 Logical model.

and congestion price values for all virtual links in the virtual network. There are three networks used to connect all elements of system as depicted in Fig. 3. Controller network (172.16.0.x) is used for controllers on PC4 to send control signals to switches PC1-3. Host control network (192.168.0.x) is used by the host manager to get information from end-hosts and controllers as well as to send control signals to end-hosts. The main network is used for end-hosts in a virtual network to send traffics to each other. Switches are connected to the first controller (named ENG) via tcp port:7000 and the second controller (named SCI) via tcp port:9000.

4.3 Virtual-Network Centralized Algorithm Implementation

We have implemented the throughput-sensitive centralized algorithm on each host manager using Python CVXOPT [15] with the objective function of maximizing the overall network throughput, meanwhile keeps the link capacity uncongested as follows:

$$\mathtt{max}: \sum_i w_i^{(k)} \log\left(\sum_j z_j^{i(k)}\right) - q \sum_{l\in L_k} \exp\left(u_l^{(k)}\right) \tag{13}$$

where the utilization $u_l^{(k)}$ of virtual link l can be calculated from $u_l^{(k)} = \frac{\mathbf{H^{(k)}} * \mathbf{z^{(k)}}}{y_l^{(k)}}$ and q serves as a weighting coefficient for regulating the utilization of virtual links. This objective function is different from [6] in that we add weight $w_i^{(k)}$ to each demand for controlling the network resources provided to each demand. This weight is calculated by $w_i^{(k)} = \frac{r_i^{(k)}}{\sum_i r_i^{(k)}}$, where $r_i^{(k)}$ is the demand i of virtual network k. End-hosts TA, TB periodically send the requested demand to the host manager which calculates the optimal flow rate for each end-host to send its traffic through each path. Upon receiving this optimal sending rate, end-hosts then adjust their outbound flow rate. We use Scapy [14] to create the packets with the wanted VLAN IDs to split traffic to different paths. For example, if end-host TA receives the optimal rate of (100, 40), it knows that it should send the traffic with the rate of 100 packets/s on VLAN 2 and 40 packets/s on VLAN 3. The end-host uses Scapy to create and send 100 packets with VLAN-ID = 2 as well as 40 packets with VLAN-ID = 3 to switch PC1. Then, at switch PC1, the packets with VLAN-ID = 2 are forwarded through the upper path with two hops and the packets with VLAN-ID = 3 are forwarded through the lower path with one hop.

4.4 Substrate-Network Distributed Algorithm Implementation

We have implemented our substrate algorithm on each OpenFlow switch. After each time a host manager completes calculating the optimal rate for all end-hosts, the host manager sends the newly updated congestion price to the OpenFlow switches. Each OpenFlow switch on receiving the congestion price calculates the new capacity of its virtual links by using the equation that we have derived in (11). The new virtual link capacity information is then updated to the host manager, which in turn uses this information in its capacity constraints to calculate the optimal flow rate for end-hosts in the next optimization step.

4.5 Simulator

Since our physical testbed is restricted to 7 PCs with 3 switching nodes, we have implemented our algorithm in MATLAB environment for two nodes topology and Abilene topology [6] to test and compare our algorithm with the previous approach in larger topology. We not only use throughput-sensitive utility function for both virtual networks as in testbed implementation but also add the delay-sensitive utility function which tries to minimize the total delay of network as follows:

$$\mathtt{min}: \sum_i \sum_j z_j^{i(1)} \sum_l H_{lj}^{i(1)}(p_l + f(u_l^{(1)})) \tag{14}$$

In our testbed implementation, with two throughput-sensitive utility functions, congestion price alone is enough to reflect the demand for more bandwidth of each virtual link. But in our simulation, the throughput-sensitive utility function and delay-sensitive utility function are so different in the form, we have added $\frac{\partial}{\partial y_l^{(k)}} U_l^{(k)}$ to congestion price λ_l^* to better reflect the demand for bandwidth than the congestion price alone.

5. Evaluation

5.1 Simulation Results

Figure 4 compares the time to convergence between our proposed algorithm and Davinci algorithm [6] in two nodes topology. The upper graph of Fig. 4 shows that with the same setting of $q = 0.5, w1 = 1, w2 = 10^5$ $\alpha = .2$ and $r_1^{(1)} = 110$ Mbps, our proposed algorithm needs nearly the same number of outer iterations, which are the number of iterations to solve the master problem (3) distributedly, as in

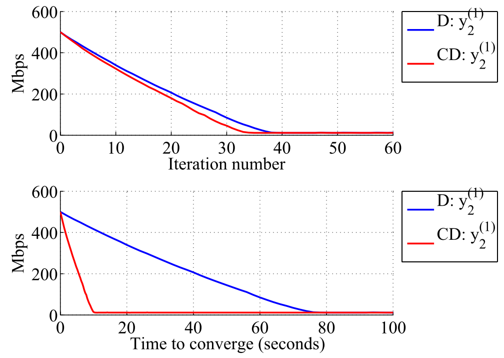


Fig. 4 Comparing with Davinci in two nodes topology.

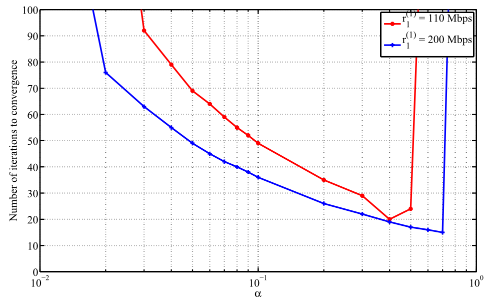


Fig. 5 Stepsize α vs. convergence in Abilene topology.

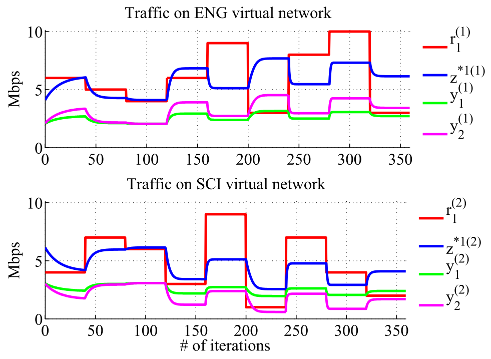


Fig. 6 Traffic on ENG and SCI network.

[6]. But, as showed in the lower graph, our algorithm needs much shorter time to converge than the one in [6]. The reason for this phenomenon is that for each outer iteration we need maximum 0.2 seconds to solve optimization problems on two virtual networks centrally but [6] needs 2 seconds to solve them distributedly. The more time needed for [6] to solve problem at each iteration make it converge slower than our proposed algorithm.

Figure 5 presents the number of iterations to convergence of our algorithm in the Abilene topology. From the figure, the algorithm converges very slow when stepsize α is so small and diverses when stepsize α is large. The reason is that when α is large, the searching point will jump through optimal point after each iteration. Also from the figure, the graph with demand $r_1^{(1)} = 200$ Mbps on delay sensitive network converges faster than the graph with demand $r_1^{(1)} = 110$ Mbps. The reason of this phenomenon is that the initial capacity for all virtual links of all virtual networks are set to be 500 Mbps which is the half of each substrate link capacity. The initial virtual link capacity vector is the initial point for our optimization algorithm. This initial point is nearer to the solution of demand 200 Mbps than the solution of demand 110 Mbps. So, the algorithm with input demand of 200 Mbps converges faster than the algorithm with input demand of 110 Mbps.

5.2 Testbed Evaluation Results

Figure 6 depicts the dynamics of actual traffic on ENG virtual network (upper graph) and SCI virtual network (lower

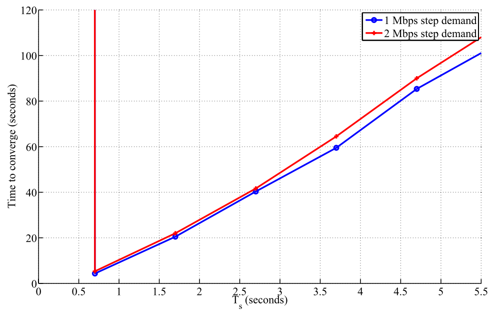


Fig. 7 Time to converge vs. T_s.

graph). This result is based on the demands which are changed randomly every 40 optimization iterations and α which is 0.05 at the beginning and being added by 0.05 each time the demand changes. The substrate link capacity is set to 5 Mbps. From the figure, the convergence rate increases after each time demand changes. But there are some periods such as 120, 200, 320, the convergence rate decreases even with higher α. This phenomenon is caused by the large changing in demand. Our algorithm uses the optimal points from previous optimization period to be the initial point for the next optimization period. If the initial point is so far from the demand, our algorithm needs more iterations to converge.

Figure 7 depicts the virtual link updated period T_s and the time to convergence in 2 demand patterns which are: increase with 1 Mbps step and 2 Mbps step. From the figure, if the virtual link update period is smaller than the maximum time to solve the optimization problem in each virtual network added with the communication time between end hosts with HostManager of that virtual network which is equal 0.7 in our case, our algorithm will need very long time to converge. If $T_s \geq 0.7$ second, the virtual network optimization problem is guaranteed to be solved completely before the updated time of our proposed algorithm (11) and our algorithm is converged. When our proposed algorithm converge, the time to convergence is quite linear with the iteration period (T_s). The reason is that the number of iterations needed to solve the master problem is nearly constant. The linear increase of time to converge is only caused by the increase in the updated time period T_s.

Figure 8 shows how virtual capacity of each virtual network adapts when there is a link-down event from iteration 36 – 148. At iteration 36, when the physical link VLAN 3 is down, the capacity which was allocated to both virtual networks ENG ($y_2^{(1)}$) and SCI ($y_2^{(2)}$) from physical link VLAN 3 is vanished. The capacity allocated to virtual network ENG ($y_1^{(1)}$) and SCI ($y_1^{(2)}$) from physical link VLAN 2 is adjusted in around 20 iterations by our updated Eq. (11) to converge to the new optimal point that the allocated virtual link capacity are proportional to the demands from both virtual networks. When the physical link of VLAN 3 is up again at iteration 148, our updated algorithm immediately allocates the equal bandwidth 3.5 Mbps to each virtual link hosted on the physical link VLAN 3. This initial allocation

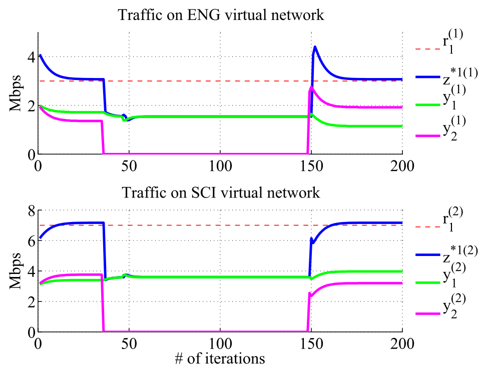


Fig. 8 Link recovery.

phenomenon caused by fair allocation of Eq. (9). After the initial allocation, our algorithm gradually adjusts the allocated capacity to the new optimal point which allocates total 3 Mbps of traffic for the virtual network ENG and 7 Mbps of traffic for the virtual network SCI.

6. Conclusion

In this paper, we have derived an algorithm to dynamically allocate the capacity to virtual network links for maximizing the aggregated utility. With the cloud resident data center setting in mind, we have used the centralized algorithm to implement in each virtual network to be suitable with OpenFlow centralized controller. At the substrate network, we have derived a new updated algorithm to make use of congestion price which is calculated from the centralized controllers. We have implemented our algorithm in MATLAB as well as testbed environment and the results show that our algorithm converge to the optimality faster than the previous approach so it can use physical resource more effectively.

To make any optimization algorithm work in the real world, we need to tune many parameters of our algorithm. The convergence time of our algorithm is based heavily on the convergence time of each centralized algorithm on each virtual network. In the future, we intend to improve the convergence rate of virtual network algorithm by precalculating the optimization problem in virtual networks with all the input demands and make a mapping table between traffic demands and primal/dual solutions. When there is a demand come to the network, the host manager only needs to find the optimal results from the mapping table. Future results will be reported accordingly.

References

[1] E. Keller, D. Drutskoy, J. Szefer, and J. Rexford, "Cloud resident data center," http://www.cs.princeton.edu/research/techreps/TR-914-11

[2] "Amazon virtual private cloud (amazon vpc)," http://aws.amazon.com/vpc/, 2011.

[3] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat, "Hedera: Dynamic flow scheduling for data center networks," Proc. 7th USENIX Conference on Networked Systems Design and Implementation, p.19, 2010.

[4] M. Alizadeh, A. Greenberg, D. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan, "Data center tcp (dctcp)," ACM SIGCOMM Computer Communication Review, pp.63–74, 2010.

[5] K. Ousterhout and J. Rexford, "Reef: A reactivate, efficient, and flexible system for internet traffic management," 2011.

[6] J. He, R. Zhang-Shen, Y. Li, C.Y. Lee, J. Rexford, and M. Chiang, "Davinci: Dynamically adaptive virtual networks for a customized internet," Proc. 2008 ACM CoNEXT Conference, CoNEXT'08, pp.15:1–15:12, New York, NY, USA, 2008.

[7] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: Enabling innovation in campus networks," ACM SIGCOMM Computer Communication Review, vol.38, no.2, pp.69–74, 2008.

[8] R. Sherwood, G. Gibb, K. Yap, G. Appenzeller, M. Casado, N. McKeown, and G. Parulkar, "Flowvisor: A network virtualization layer," OpenFlow Switch Consortium, Tech. Rep. 2009.

[9] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker, "Nox: Towards an operating system for networks," ACM SIGCOMM Computer Communication Review, vol.38, no.3, pp.105–110, 2008.

[10] D. Eppstein, "Finding the k shortest paths," SIAM J. Comput., vol.28, no.2, pp.652–673, 1998.

[11] M. Pióro and D. Medhi, Routing, flow, and capacity design in communication and computer networks, Elsevier/Morgan Kaufmann, 2004.

[12] D. Palomar and M. Chiang, "A tutorial on decomposition methods for network utility maximization," IEEE J. Sel. Areas Commun., vol.24, no.8, pp.1439–1451, 2006.

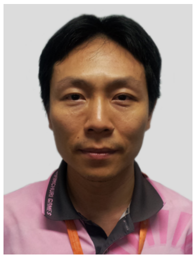
[13] S. Boyd and L. Vandenberghe, Convex optimization, Cambridge Univ Pr, 2004.

[14] "Scapy." http://www.secdev.org/projects/scapy

[15] "Cvxopt: Python software for convex optimization." http://abel.ee.ucla.edu/cvxopt/index.html

Tri Trinh received Bachelor as well as Master degree from Hanoi University of Science and Technology, Vietnam in 2004 and 2006 respectively. He is now a Ph.D. student in the Department of Electrical Engineering, Chulalongkorn Unversity since 2009. In 2011, he was an exchanged researcher in Esaki's lab, The University of Tokyo. His research interest is Network Virtualization, Cloud Resident Data Center, OpenFlow, Traffic Engineering, Network Optimization.

Hiroshi Esaki received the Ph.D. degree from the University of Tokyo, Tokyo, Japan, in 1998. In 1987, he joined the Research and Development Center, Toshiba Corporation. From 1990 to 1991, he was with the Applied Research Laboratory, Bell-Core Inc., NJ, as a Residential Researcher. From 1994 to 1996, he was with the Center for Telecommunication Research, Columbia University, New York. Since 1998, he has served as a Professor at the University of Tokyo, and as a board member of WIDE Project. Currently, he is the Executive Director of IPv6 promotion council, Vice President of JPNIC, IPv6 Forum Fellow, Director of Green University of Tokyo Project, and Director of WIDE Project.



Chaodit Aswakul received the B.Eng. degree with 1st class honour in Electrical Engineering from Chulalongkorn University, Thailand, in 1994. From 1995 to 2000, he received the Ananda Mahidol Foundation scholarship to study for Ph.D. at the Imperial College of Science, Technology and Medicine, University of London, U.K. After getting Ph.D. in communications networking, he returned to Thailand and is currently an assistant professor at the Department of Electrical Engineering, Chulalongkorn University. In 2008–2009, he has also served as the manager of the Next Generation Network Testing/Trial Use Project in Phuket of the National Telecommunications Commission of Thailand. He is the cofounder of the university's Network Research Group with focuses on researching in the quality of service and reliability issues of both communication and transportation network systems.