STATIC AND DYNAMIC RESOURCE ALLOCATION IN NETWORK VIRTUALIZATION FOR MULTI-TENANT DATA CENTER NETWORK

Mr. Trinh Minh Tri

A Dissertation Submitted in Partial Fulfillment of the Requirements
for the Degree of Doctor of Philosophy Program in Electrical Engineering
Department of Electrical Engineering
Faculty of Engineering
Chulalongkorn University
Academic Year 2012

การจัดสรรทรัพยากรแบบสถิต และแบบพลวัตในการสร้างโครงข่ายเสมือนสำหรับโครงข่ายศูนย์ข้อมูลที่มีผู้เช่าหลายราย

นาย ทริน มิน ทริ

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรดุษฎีบัณฑิต
สาขาวิชาวิศวกรรมไฟฟ้า ภาควิชาวิศวกรรมไฟฟ้า
คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย
ปีการศึกษา 2555

Thesis Title	STATIC AND DYNAMIC RESOURCE ALLOCATION IN NETWORK VIRTUALIZATION FOR MULTI-TENANT DATA CENTER NETWORK
By	Mr. Trinh Minh Tri
Field of Study	Electrical Engineering
Thesis Advisor	Assistant Professor Chaodit Aswakul, Ph.D.
Thesis Co-advisor	Professor Hiroshi Esaki, Ph.D.

Accepted by the Faculty of Engineering, Chulalongkorn University in Partial Fulfillment of the Requirements for the Doctoral Degree

. . . . . Dean of the Faculty of Engineering
(Associate Professor Boonsom Lerdhirunwong, Dr.Ing.)

THESIS COMMITTEE

. . . . . Chairman
(Associate Professor Watit Benjapolakul, D.Eng.)

. . . . . Thesis Advisor
(Assistant Professor Chaodit Aswakul, Ph.D.)

. . . . . Thesis Co-advisor
(Professor Hiroshi Esaki, Ph.D.)

. . . . . Examiner
(Assistant Professor Chaiyachet Saivichit, Ph.D.)

. . . . . External Examiner
(Associate Professor Poompat Saengudomlert, Ph.D.)

. . . . . External Examiner
(Patrachart Komolkiti , Ph.D.)

ทริน มิน ทริ: การจัดสรรทรัพยากรแบบสถิต และแบบพลวัตในการสร้างโครงข่ายเสมือนสำหรับโครงข่ายศูนย์ข้อมูลที่มีผู้เช่าหลายราย (STATIC AND DYNAMIC RESOURCE ALLOCATION IN NETWORK VIRTUALIZATION FOR MULTI-TENANT DATA CENTER NETWORK)
อ.ที่ปรึกษาวิทยานิพนธ์หลัก: ผศ.ดร.เชาวน์ดิศ อัศวกุล, 91 หน้า.

ทิศทางหลักสองทางสำหรับการจัดสรรทรัพยากรโครงข่ายเสมือนในโครงข่ายศูนย์ข้อมูลที่มีผู้เช่าหลายรายได้แก่แนวทางแบบสถิต และแนวทางแบบพลวัต แนวทางแบบสถิตหาทรัพยากรที่เหมาะที่สุดของโครงข่ายซับสเตรตสำหรับการเป็นแม่ข่ายให้กับโครงข่ายเสมือนแต่ละโครงข่าย และตั้งค่าทรัพยากรต่าง ๆ ไว้ถาวรตลอดระยะเวลาของการใช้งานโครงข่ายเสมือนนั้น แนวทางแบบพลวัตสามารถปรับค่าทรัพยากรต่าง ๆ ได้แบบรายคาบสำหรับโครงข่ายเสมือนเพื่อที่จะให้ได้ค่าสมรรถนะโครงข่ายโดยรวมสูงสุด อย่างไรก็ตามวิธีซึ่งมีอยู่ตามแนวทางทั้งสองแนวทางนั้นไม่สามารถทำงานได้อย่างมีประสิทธิผลในการสร้างโครงข่ายเสมือนสำหรับสถานการณ์ต่าง ๆ ของโครงข่ายศูนย์ข้อมูลที่มีผู้เช่าหลายราย วิธีต่าง ๆ ของการจัดสรรแบบสถิตกำหนดบริการในรูปแบบได้รูปแบบเดียวเพื่อให้กรรมสิทธิ์การใช้งานแต่ผู้เดียวแก่การขอใช้งานแต่ละครั้ง วิธีต่าง ๆ ของการจัดสรรแบบพลวัตไม่เหมาะมที่สุดสำหรับศูนย์ข้อมูลที่อยู่ในคลาวด์ซึ่งเป็นส่วนสำคัญของโครงข่ายศูนย์ข้อมูลที่มีผู้เช่าหลายราย วิทยานิพนธ์ฉบับนี้มีเป้าหมายที่จะใช้เครื่องมือการหาค่าเหมาะที่สุดเพื่อแก้ปัญหาในแนวทางทั้งสองแนวดังต่อไปนี้

ในแนวทางแรกวิทยานิพนธ์นี้นำเสนอการวิเคราะห์แนวคิดของการจองมากกว่าที่มีอย่างระมัดระวังซึ่งมีความยืดหยุ่นของระดับสภาพพร้อมใช้งานในข้อตกลงระดับการให้บริการที่ให้กับผู้ใช้ต่าง ๆ บนพื้นฐานของแนวทางนี้ผู้เช่าโครงข่ายเสมือนได้รับบริการที่เหมาะที่สุดกับระดับการรับประกันแบนวิดท์ให้แบบชั่วคราวที่สามารถยอมรับได้โดยผู้ใช้นั้นซึ่งช่วยในการประหยัดค่าใช้จ่ายของผู้ใช้และเพิ่มความสามารถในการทำกำไรของผู้ให้บริการ นอกจากนี้เพื่อค้นหาเส้นทางของทราฟฟิกที่เป็นผลมาจากแนวคิดของการจองมากกว่าที่มีอย่างระมัดระวังในโครงข่ายซับสเตรต วิทยานิพนธ์นี้ได้สร้างข้อปัญหากำหนดการเชิงจำนวนเต็มผสมเพื่อลดค่าใช้จ่ายของโครงข่ายเสมือน รายงานผลการคำนวณแสดงให้เห็นว่าวิธีการที่เสนอสามารถลดค่าใช้จ่ายในการดำเนินการของโครงข่ายเสมือนไปพร้อมกับการรับประกันคุณภาพบริการที่ต้องการได้

ในแนวทางที่สองโครงข่ายศูนย์ข้อมูลที่อยู่ในคลาวด์ได้รับการพิจารณา ซึ่งวิทยานิพนธ์นี้ได้มุ่งเน้นที่แพลตฟอร์มโอเพนโฟลวเป็นองค์ประกอบหลักเพื่อเพิ่มความยืดหยุ่นให้กับผู้ใช้งานศูนย์ข้อมูลแบบเสมือน ในสถานการณ์นี้โครงข่ายเสมือนต่าง ๆ ในทางปฏิบัติจะมีขนาดเล็ก ดังนั้นวิทยานิพนธ์นี้จึงนำเสนอขั้นตอนวิธีแบบรวมศูนย์เพื่อใช้ในการควบคุมเส้นทาง และปริมาณของทราฟฟิกในโครงข่ายเสมือนแต่ละโครงข่าย ในขณะเดียวกันเนื่องจากโครงข่ายซับสเตรตที่มีขนาดใหญ่ดังนั้นขั้นตอนวิธีแบบกระจายได้ถูกใช้เพื่อปรับค่าแบนวิดท์แบบรายคาบสำหรับข่ายเชื่อมโยงต่าง ๆ ของโครงข่ายเสมือน เพื่อประเมินผลวิธีที่นำเสนอนี้ การตรวจสอบต่าง ๆ ที่ได้ดำเนินการในวิทยานิพนธ์ครอบคลุมถึงการวิเคราะห์ทางทฤษฎี การจำลองระบบด้วยโปรแกรมคอมพิวเตอร์ และการวัดผลต่าง ๆ ในระบบทดสอบขนาดเล็กของศูนย์ข้อมูลที่อยู่ในคลาวด์ซึ่งใช้โอเพนโฟลวเป็นพื้นฐาน ผลการทดลองต่าง ๆ ที่ได้นั้นแสดงให้เห็นว่าขั้นตอนวิธีใหม่มีเวลาคำนวณเพื่อลู่เข้าสู่ผลลัพธ์ที่เร็วขึ้น ใช้การคำนวณที่ง่ายขึ้น และสามารถใช้ข้อมูลป้อนกลับได้ดีกว่าขั้นตอนวิธีต่าง ๆ ที่มีการนำเสนอในอดีต

ภาควิชา วิศวกรรมไฟฟ้า ลายมือชื่อนิสิต .....

สาขาวิชา วิศวกรรมไฟฟ้า ลายมือชื่อ อ.ที่ปรึกษาวิทยานิพนธ์หลัก .....

ปีการศึกษา 2555

##4871872421: MAJOR ELECTRICAL ENGINEERING
KEY WORDS: NETWORK VIRTUALIZATION / OPENFLOW / RESOURCE ALLOCATION/ OPTIMIZATION

TRINH MINH TRI: STATIC AND DYNAMIC RESOURCE ALLOCATION IN NETWORK VIRTUALIZATION FOR MULTI-TENANT DATA CENTER NETWORK
THESIS ADVISOR : ASST. PROF. CHAODIT ASWAKUL, Ph.D., 91 pp.

The two main directions for virtual network resource allocation in multi-tenant data center network are static and dynamic approaches. The static approach finds the optimal resources of substrate network for hosting each virtual network and keeps the resources unchanged for the whole lifetime of the virtual network. The dynamic approach can adjust the resources periodically for the virtual network to maximize the total network performance. However, the existing methods in both approaches work ineffectively in the network virtualization for multi-tenant data center network situations. The static allocation methods provide only exclusive service that allocates the full availability to each request. The dynamic allocation methods are not optimized for cloud resident data center, which is an important part of multi-tenant data center network. This dissertation aims to use optimization tools to solve the problem in both approaches as follows.

In the first approach, this dissertation has proposed the analysis of careful overbooking concept, which uses the flexible level of availability in the service level agreement provided to users. Based on this approach, virtual network subscribers are provided with a service that is most suitable with their tolerability to utilize the soft-guaranteed bandwidth. This helps saving the cost of subscribers and increasing the profitability of provider. Furthermore, to route the traffic resultant from the careful overbooking concept through a substrate network, this thesis has formulated the mixed integer programming problem to minimize the virtual network cost. The reported results demonstrate that the proposed method can reduce the operating cost of virtual network while guaranteeing the required quality of service.

In the second approach, the cloud resident data center network has been considered. The dissertation emphasis is on the OpenFlow platform as the key element to provide the flexibilities to virtual data center customers. In this situation, since the practical virtual networks are of small size, this thesis has proposed to centralize the algorithm to control the amount of traffics on routes in each virtual network. At the same time, due to the big size of substrate network, the distributed algorithm is used to adjust bandwidth periodically for virtual network's links. To evaluate the proposed method, investigations carried out in this thesis include the theoretical analysis, computer simulation, and small OpenFlow-based cloud resident data center testbed measurements. The obtainable results show that the new algorithm has faster solution-convergence time, simpler calculation, and can make the better use of feedback information from virtual networks than the previous algorithms.

Department: Electrical Engineering Student's Signature .....
Field of study: Electrical Engineering Advisor's Signature .....
Academic year: 2012

Acknowledgements

First and foremost, I would like to express my sincere thanks to my advisor Asst. Prof. Dr. Chaodit Aswakul for introducing me to the world of network research. His intricacy, resiliency and sincerity not only allow me to work with ease, but also encourage me to fearlessly pursue the research ideas. Being a student of his is one of the luckiest experiences in my life. Furthermore, I am greatly thankful to my co-advisor Prof. Hiroshi Esaki for keeping remind me to find the real problems from engineers. His strict and commitment let me finally understand that a good research should not only be intellectually exciting, but also practical and useful. I would like to express my thanks to Asst. Prof. Dr. Chaiyachet Saivichit for sharing his invaluable research experience during the hard time of my graduate study. It is honour to me to have Assoc. Prof. Dr. Watit Benjapolakul as a chair person of my thesis committee. I would also like to thank Assoc. Prof. Dr. Poompat Saengudormlert for his helpful suggestions and comments. My warmest gratitude thanks to Dr. Patrachart Komolkiti, my senior and now my committee. His kindness with people, helpfulness with colleges and profession in work are always the mirror for me to follow.

Second, I am grateful to AUN/SEEDNET for not only providing me the scholarship to study Ph.D, but also taking care of me in my daily life.

Third, I would like to thank many of my colleagues in the research lab of Chulalongkorn University. Many thanks to my doctoral colleagues Dr. Kalika, Dr. My, Pitipong, Teerapol, Kamontep, Sarisom, Kittisak, Robithol Anur, Omer for their helpful discussions. Special thanks to Dr. Kalika who brought me around Tokyo to present and talk with so many OpenFlow experts. I would like to thank the master's degree students in my lab too: Suwatchai, Nuttida, Boby, Snow, Fasai, Singha, A, Game, Chanthan, Katan, Pae, Bao, Dew and many others that I will never forget.

I benefited a lot from my two internships to the University of Tokyo. Many thanks to Asst. Prof. Hideya Ochiai for guiding me through many difficulties when implementing OpenFlow. Special thanks to Ryota Kosaka for his kindness and helpfulness. I would like to thank Thomas Silverston for his guidance in research. I would like to thank Jorge Otiz for sharing how to research. I would like to thank Dr. Yusuke, Dr. Hirochika, Dr. Kanaumi, Kaveevivitchai, Lertluck, Kawaguchi, Motodate, Luchiano, Higashiura, Kinoshita. I would like to thank Dr. Keiichi Shima for his insightful comment about OpenFlow.

Last, but not least, I owe my deepest gratitude to my family members. Thanks dad for always asking me to speed-up my research. Thanks mom for always asking me to take care my health. Thanks my sister and my brother in law for taking care of my parents when I am pursing Ph.D.

Contents

Chapter Page

List of Figures

CHAPTER I

INTRODUCTION

Network virtualization has been recently introduced as a promising proposal for a multi-tenant data center. It can provide additional flexibility and enhance the performance isolation among tenants. However, the tradeoff for the flexibility is the additional complexity of resource allocation algorithms which help to use data center resources efficiently. The new network constraints and new network management model should be taken into account when deriving the new resource allocation algorithms. This dissertation aims to use the optimization theory to derive the new resource allocation algorithms in network virtualization for multi-tenant data center network.

1.1 Network virtualization for multi-tenant data center

A multi-tenant data center as shown in Figure 1.1 is a data center with the ability of accommodating many tenants e.g. different enterprises, in a shared manner. Up to the degree of performance isolation, a multi-tenant data center can provide server hosting service or IaaS service. The server hosting service is used to serve big customers such as Google and Nasdaq cloud to extend their data center to remote areas where their own physical data centers are unreachable. The big customers need high performance isolation so they usually bring their own physical servers to locate in the accommodated data center and rent the hosting space, air conditioners, power and networks to operate their servers. The IaaS service is used to serve the smaller customers e.g. Mendelay, Instagram, Newsweek, WashingtonPost, Photomedia. These customers do not need high performance isolation, but need fast and convenient way to create and expand their networks. Thanks for computer virtualization technologies [3], many virtual machines can be created on a physical machine. The virtual machines can be networked and provided as IaaS to these small customers.

Each tenant renting service in a multi-tenant data center has its own performance requirements. For instance, Nasdaq tenant with its stock exchange service requires a low end-to-end delay to synchronize the stock exchange index among its back-end servers; Google

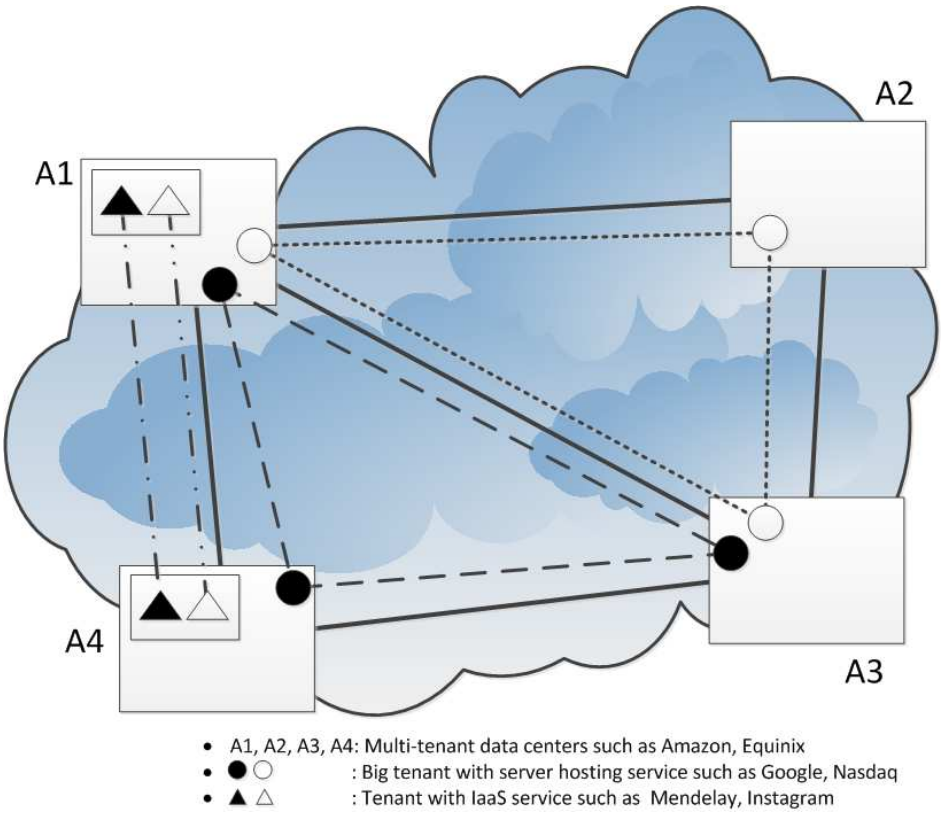

Figure 1.1: Multi-tenant data center

tenant with its searching service requires a high throughput to synchronize the searching results among its back-end servers.

In the current state of art, traditional technologies, such as closed network boxes and TCP/IP protocol [4] are still used to network the components in the multi-tenant data center. The traditional technologies only provide a simple sharing environment for hosted services, but hardly do they provide the service isolation. It is hard for a provider to guarantee QoS in a sharing environment. For example, there are two services from two different tenants, one requires throughput-sensitive traffic the other requires delay-sensitive traffic. Traffic from both applications flows through the same physical link of a multi-tenant data center. To support the throughput-sensitive traffic, the queuing system in the physical link needs to create a long queue. To support the delay-sensitive service, the queuing system in the physical link needs to create a short queue. However, it is difficult for the queuing system in the physical link to control the queue length for supporting both delay-sensitive and throughput-sensitive service.

To facilitate the performance isolation and guarantee the quality of service in the traditional network, researchers have already introduced many integrated modules such as loca-

tion, classification, authentication, encryption, roaming, identification, reservation and differentiation [4]. These integrated modules only act as the patches for the current weaknesses of the traditional network with the price of making it heavy, complicated and expensive in operating as well as maintaining. The other solution from patching many extra integrated modules for the current network weaknesses is to build the new model of network virtualization.

Network virtualization is a model of building many heterogeneous virtual networks on a shared physical substrate network. This model promises to provide stronger performance isolation and more flexibility to virtual networks than the ones of traditional network. The virtual nodes are isolated from each other by computer virtualization technologies e.g. Xen [3], KVM [5]. The virtual link bandwidth isolation can be developed from the existing technologies such as Diffserve and Intserve [4]. The additional flexibility in network virtualization let the tenants be able to implement their own network protocols which have been customized to best support the tenants' services without affecting performance of others. For example the Google tenant can implement the long queue system to maximize its network throughput. Mean while, the Nasdaq tenant can implement the short queue system in their network to minimize its network delay.

There are many similarities between multi-tenant data center and network virtualization model. The landlord, who owns as well as manages the data center and allocate services to customers, is equivalent to Infrastructure Provider (InP), who owns as well as manages the substrate network and allocate services to customers. The tenants who rent a part of multi-tenant data center to implement their services are equivalent to Service Providers (SPs) who rent virtual networks from the InP. The network part rented by a tenant is equivalent to a virtual network. A physical data center is equivalent to a substrate network.

Network virtualization is proposed from many successes of previous virtualization technologies. MPLS-VPN [6] virtualizes the routing tables in the ISP routers by giving each VPN network a dedicated virtual routing table. VLAN [7] virtualizes the network's link layer to create many separated virtual networks on the same physical network. WDM [8] virtualizes the physical network by using wavelength separation. An overlay network [8] is a virtual network of physical nodes located in customer sites and virtual links hosting on physical paths of carrier's network.

However, there are three differences between network virtualization and previous vir-

tualization technologies such as the overlay [9]. The first difference is in the controlling ability of tenants in the provider's network. The overlay networks rent the end-to-end virtual links from the carrier's network. The overlay customers do not know hence are unable to control how their traffic flows in the carrier's network. In other words, overlay customers see the carrier's network as a black box through which to push the traffic. On the contrary, SPs in network virtualization architecture can see into their substrate's portion, and can control how the traffic is forwarded, dropped, filtered, routed in the InP's network. The second difference between the network virtualization and previous virtualization technologies is that overlay network cannot rent the nodes in the substrate network. Instead, the network virtualization architecture lets InP create many virtual nodes on one physical node, and lease it to SPs. The final difference is that the isolation between virtual networks in network virtualization is stronger than the isolation in the overlay network which hosts virtual links on the substrate network paths in a shared manner.

Network virtualization relieves the substrate network, and provides benefits to joined parties. The extra feature modules of the network such as location, classification, authentication, encryption, roaming and identification are unloaded from the infrastructure network. The infrastructure network becomes easier to manage, lighter, smoother and more scalable than before implementing virtualization. The SPs can selectively as well as exclusively implement these modules in SPs' virtual networks to support their services [10]. The InPs gain benefits from network by extending their class of customers from end users to SPs. The SPs with the help of network virtualization do not need to build their own physical infrastructure but instead rent the network resources such as link bandwidths and node processes from the InP. This mechanism makes service provision easy, and breaks down the barrier to network service business. As well, users are promised to have the high quality and cheap price services provided by the competing SPs.

The main barrier for achieving network virtualization in a multi-tenant data center is the collocation of the data plane and the forwarding plane in the same network box e.g. switch, router, access point [11]. This collocation dismisses the traffic control ability of tenants in their rented portions, but gives all the traffic control power to data center's landlord. This unbalance of function design makes the multi-tenant data center network inflexible. Furthermore, the collocation prevents network innovations from being tested in the sufficiently realistic settings to gain enough confidence before the implementation within commercial

networks. The lacking of innovations implemented in the multi-tenant data center network make it “ossified".

OpenFlow [12] has been proposed to overcome the collocation problem by creating a standard interface for data plane, defining a controller and a protocol to communicate between controller and the data plane’s standard interface. OpenFlow defines a common interface for forwarding tables. As OpenFlow, all kinds of switches has common fields in their forwarding tables such as *input port*, *output port*, *source MAC address*, *destination MAC address*, *source IP*, *destination IP*, *source port* and *destination port*. These common fields can be exploited to build a common programming interface for OpenFlow switches. This process is equivalent to computer operating system building abstraction layer for computing hardware. The OpenFlow also defines a controller which provides Application Programming Interface (API) and facilitates the implementation of control plane’s applications. OpenFlow protocol is also defined for a controller (control plane) to remotely manipulates common interfaces in the switch forwarding tables (data plane). This protocol can be easily implemented in most of current switches without much changing in the switch hardware.

The OpenFlow protocol has many types of messages [13]. The Controller-to-Switch messages are sent from a controller to control switches. Hand shake, switch configuration, flow table configuration, modify state, queue configuration, read state, packet-out, barrier messages are the messages of this type. Asynchronous messages are sent from one side, OpenFlow controller or OpenFlow switch, to the other side without the necessary reply. Packet-in, flow-remove, port-status and error are the messages of this type. Symmetric messages, such as hello, echo request, echo reply, and experimenter need the reply from the receiver. Besides supporting atomic actions such as add/delete/change entries in the switches, OpenFlow defines the ways to combine those actions such as pipe-line processing, action set and action list. The matching fields are very flexible with 10-tuple ranging from layer one to layer four of TCP/IP model. The working process of OpenFlow system can be described by Figure 1.2.

Solving collocation between control and data plane is very important but not enough to build the network virtualization. To enable network virtualization, it is necessary to have mechanisms to slice the substrate network resources, and to isolate them among virtual networks. FlowVisor [1], which acts as a transparent proxy between the OpenFlow switches and the Controllers, is the tool for this mission. FlowVisor slices a substrate network by di-

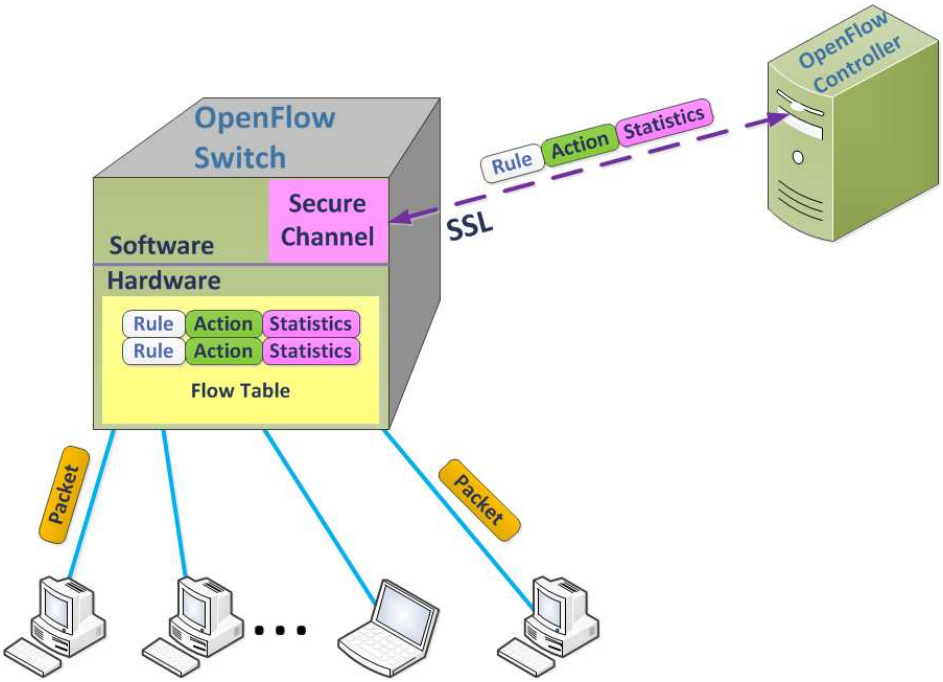

Figure 1.2: OpenFlow connection mechanism

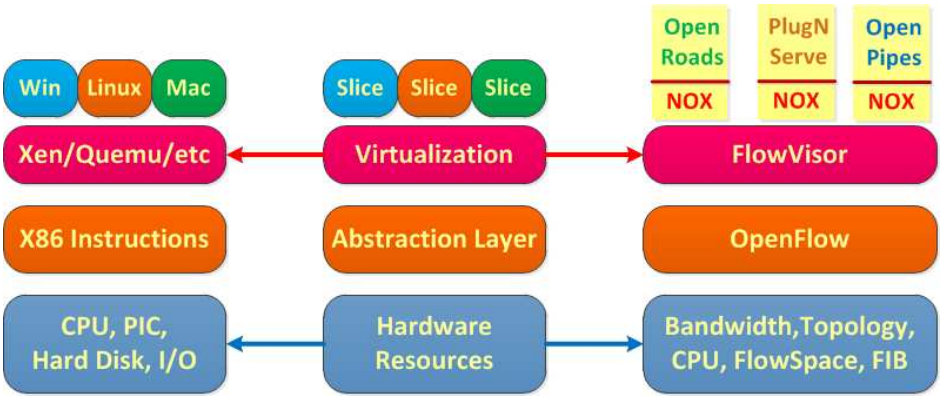

Figure 1.3: Comparison of network virtualization and computer virtualization [1]

viding its flow space into many flow subspaces for allocating to virtual networks. FlowVisor inspects all OpenFlow messages between the controller and the switches to make sure the controller can only send the message to its managed flow subspaces. Furthermore, FlowVisor also isolates five types of resources among all virtual networks: bandwidth, topology, traffic, cpu and forwarding tables. With the ability of dividing physical resources and isolating among them, FlowVisor along with OpenFlow can build the new network architecture for network virtualization. In the FlowVisor's model, the SP's controllers use OpenFlow protocol to communicate with the infrastructure network via a secure channel. The network operating system [14] installed on the controller is the same as the operating system installed on computers. The developers write applications to control the infrastructure sub-flows the same as the developers write applications on Windows® or Ubuntu®.

As shown in Figure 1.3, network virtualization architecture built from OpenFlow and FlowVisor have the same principle as computer virtualization. The computer's virtualization layer sits between underlying hardware and software. Similarly, the virtualization layer, FlowVisor, sits between the network physical resources and controllers. The computer's hardware abstraction layer can pool and slice hardware resources. This layer makes the hardware plug easily into the standard interfaces while making the operating system believe of having its own hardware. FlowVisor is a network abstraction layer for pooling and slicing all the underlying hardware resources. Multiple virtual networks can run simultaneously on top of this virtualization layer without interfering with each other. This abstraction layer has been successful in computer virtualization, so will be in the network virtualization.

Resource allocation in network virtualization refers to mechanism of allocating virtual nodes and links on physical nodes and paths, respectively. These mechanisms help not only improve the resource utilization as well as avoiding the congestion in substrate but also maximize the performance objectives of InP and SPs. For example, with VDO service, the SP can implement the delay mechanism to minimize the network delay and with the file transfer service, the SP can implement the throughput mechanism to maximize network throughput. The InP can implement the mechanism to maximize performance of all virtual networks. If network virtualization has a good resource allocation mechanism, the network virtualization resources can be used effectively in the way of best supporting all the virtual network demands. The next section will present about resource allocation approaches in network virtualization.

1.2 Resource allocation approaches in network virtualization

To maximize their profit, InPs have to use resource allocation to make most use of the expensive infrastructures [2]. However, the resource allocation in the network virtualization is more complicated than the resource allocation in the traditional network.

To optimize the allocation of bandwidth in a substrate network, InPs not only take into account the traffic demands as in the traditional network, but also the commands from guest controllers. These commands make the traffic's direction change unexpectedly, and add dynamic constraints to the optimal resource allocation problem [12]. Further, there are more ways for the substrate to deal with the congestion in the network virtualization than in the conventional networks. For example, the congestion in a virtual link does not always mean that the sources have to decrease their transmission rates. In such a situation, the other possibility is to increase virtual link's capacity [15].

Resource allocation in network virtualization assigns not only the pipe bandwidth but also the node resources. The problem of allocating virtual nodes to substrate nodes, without violating bandwidth constraints, is NP-hard and similar to the multi-way separator problem [16]. To solve such problems, three approaches have been identified: brute-force backtracking algorithm, simulated annealing algorithm and approximation algorithm.

The resource allocation in the network virtualization environment refers to the static or dynamic allocation of virtual nodes and links on physical nodes and paths accordingly. In the static approach (initial network provisioning), the allocated resources do not change during the virtual network life time. On the contrary, the dynamic approach (adaptive virtual network provisioning) adjusts the allocated resources periodically [2].

1.2.1 Problem description

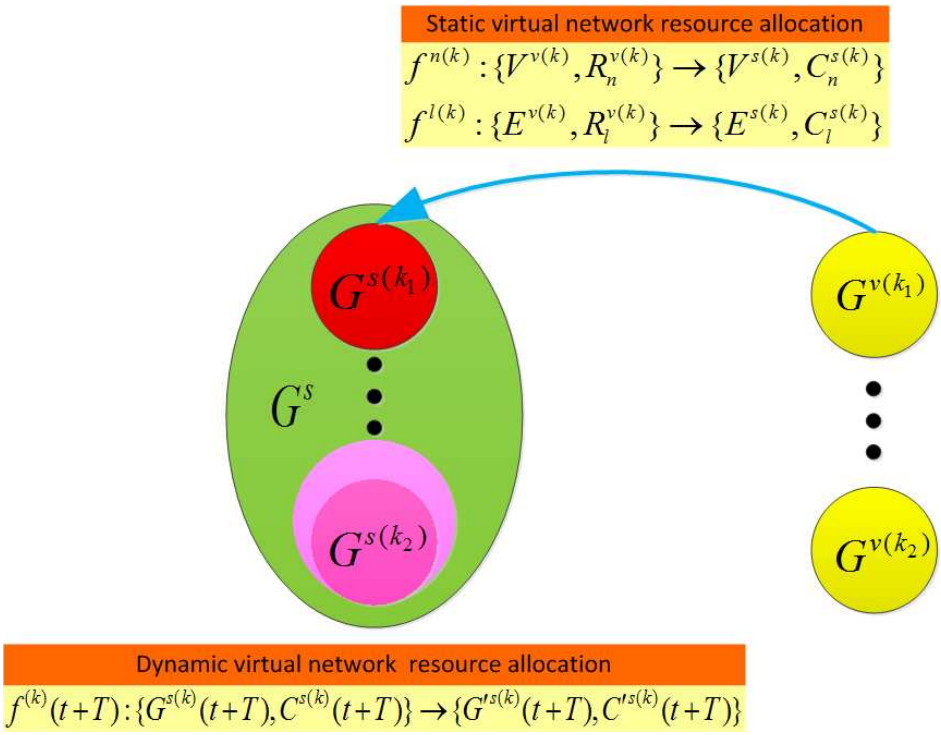

Figure 1.4: Static and dynamic virtual network resource allocation [2]

A substrate network is modelled as a graph of nodes and links $G^s(V^s, E^s, C_n^s, C_l^s)$ where V^s is the set of all substrate nodes each of which is identified by an index number n and a capacity C_n^s; and E^s is the set of all substrate links each of which is identified by an index number l and a capacity C_l^s. Note that s here stands for substrate, and is not an index number. The substrate network hosts many virtual networks, each of which is identified by the index number k. The virtual network k is modelled by $G^{v(k)} = \{V^{v(k)}, E^{v(k)}, R_n^{v(k)}, R_l^{v(k)}\}$ where $V^{v(k)}$ is the set of all virtual nodes in the virtual network k, $E^{v(k)}$ is the set of all virtual links in the virtual network k, $R_n^{v(k)}$ is a resource requirement for the virtual node n in the virtual network k, and $R_l^{v(k)}$ is a resource requirement for the virtual link l in the virtual network k. Note that v here stands for virtual, and is not an index number. As shown in Figure 1.4, the image of virtual network k, $G^{v(k)}$, on the substrate network is $G^{s(k)}$. Based on the dynamic of $G^{s(k)}$, the virtual network allocations are classified as static or dynamic.

In the static virtual network allocation, the substrate network finds out the resources to host a virtual network from the initial requirements of the virtual network. When the virtual network is hosted successfully onto the substrate network, the virtual network will

not change its resource demands. In other words, the image of virtual network k, $G^{v(k)}$, on the substrate network, $G^{s(k)}$, does not change with time t. Mathematically, the arrival of a virtual network demand k will yield the substrate network to calculate the two mappings, which are node mapping $f_n^{(k)}$:$\{V^{v(k)}, R_n^{v(k)}\} \longrightarrow \{V^{s(k)}, C_n^{s(k)}\}$ and link mapping $f_l^{(k)}$:$\{E^{v(k)}, R_l^{v(k)}\} \longrightarrow \{E^{s(k)}, C_l^{s(k)}\}$. Note that $R_n^{v(k)}$ and $R_l^{v(k)}$ here are constants for each virtual network k, and are not a function of time t. The mission of static virtual network allocation here is to define the two mappings $f_n^{(k)}$ and $f_l^{(k)}$.

Different from the static approach, the dynamic virtual network allocation calculates the substrate resources for virtual networks based on their actual resource usage. It periodically re-scales the capacity allocated to virtual nodes and links to match with their actual usage. Mathematically, assume that the resource provided to virtual network k at time t is $G^{s(k)}(t) = \{V^{s(k)}(t), E^{s(k)}(t), C_n^{s(k)}(t), C_l^{s(k)}(t)\}$. As these provided resources are different from actual usage of virtual network k, the dynamic virtual network allocation re-adjusts the resources for that virtual network after each period T by calculating the following mapping $f^{(k)}(t+T)$:$\{G^{s(k)}(t+T), C^{s(k)}(t+T)\} \longrightarrow \{G'^{s(k)}(t+T), C'^{s(k)}(t+T)\}$ where $\{G^{s(k)}(t+T), C^{s(k)}(t+T)\}$ is the prior substrate capacity allocated to virtual network k and $\{G'^{s(k)}(t+T), C'^{s(k)}(t+T)\}$ is the new re-adjusted capacity that substrate provides to virtual network k. The mission of the dynamic virtual network allocation here is to define the mapping $f^{(k)}(t+T)$.

1.2.2 Static virtual network allocation

As in Section 1.2.1, there are two approaches for virtual network resource allocation: static and dynamic approach. The static approach implies the virtual network assignment without reconfiguration [17]. In this paper, Zhu and Ammar have defined the notion of "stress of a substrate object" as the number of virtual objects hosting on a substrate object. For example, the stress of a substrate node is the number of virtual nodes hosting on the substrate node, and the stress of a substrate link is the number of virtual links hosting on the substrate link. By minimizing the highest stress on both substrate's nodes and links, the authors' optimization formulation can balance the substrate network's stress. Two heuristic methods, which find a set of optimal substrate nodes and links for each virtual network demand, are proposed to solve this optimization problem. The basic method finds the substrate nodes to add to the allocated node set. The first substrate node is chosen to add to the set if it

has the lowest stress on both that node and the directly connected links. The following substrate nodes are chosen with the lowest stress on both that node and the shortest paths connecting that node to all the substrate nodes in the set. When the set has enough nodes to host the virtual network, the algorithm uses the shortest path [18] to assign the substrate link resources for virtual links. The second method exploits the simplicity of embedding a star topology network to the substrate network. It breaks a complicated virtual network to many simple star sub-networks, then maps them onto the substrate network. The simulations show two important results. Firstly, the sub-dividing virtual network algorithm reduces the computational time. Secondly, in case of sparse virtual network connectivity, this method can reduce the highest link stress while keeping the highest node stress unchanged.

The next approach for the static virtual network allocation is presented in [19]. In their paper, to mitigate the NP-hard problem, Lu and Tunner have reduced the searching space by restricting the virtual network topology to the virtual backbone-star topology which has virtual backbone nodes connecting to virtual access nodes. The algorithm is shown in Figure 1.5, which have been taken from [2] with some adjustments. There are two advantages of this algorithm. Firstly, the substrate network does not need time for finding resources on which to map the virtual access nodes. Secondly, instead of having to search any virtual network topologies and fit them to the substrate network, the algorithm only needs to search some specific topologies of complete, ring or star from backbone network. These constraints reduce searching space, and hence make the algorithm faster than others. The simulation experiments of this algorithm show that the cost of constructing a virtual network is usually no more than 1.5 times a computed lower bound on network cost.

The next static virtual network allocation method has been proposed in [20]. In this approach, Yu et al. find the substrate nodes and links to host a virtual network demand based on the objective of maximizing the total revenue. They use the greedy method to calculate and map the virtual nodes onto substrate nodes, then find the paths to connect those substrate nodes by k-shortest path algorithm [21]. The innovation in the method is mostly in the way the authors play with the virtual link demands and the substrate paths to deal with the NP-hard problem. The algorithm allows the substrate network to split the virtual link demands over multiple substrate paths. The algorithm also employs a path migration to periodically re-optimize the utilization of the substrate network. The experimental results show that with the two new mechanisms, path splitting and path migration, the substrate network can get

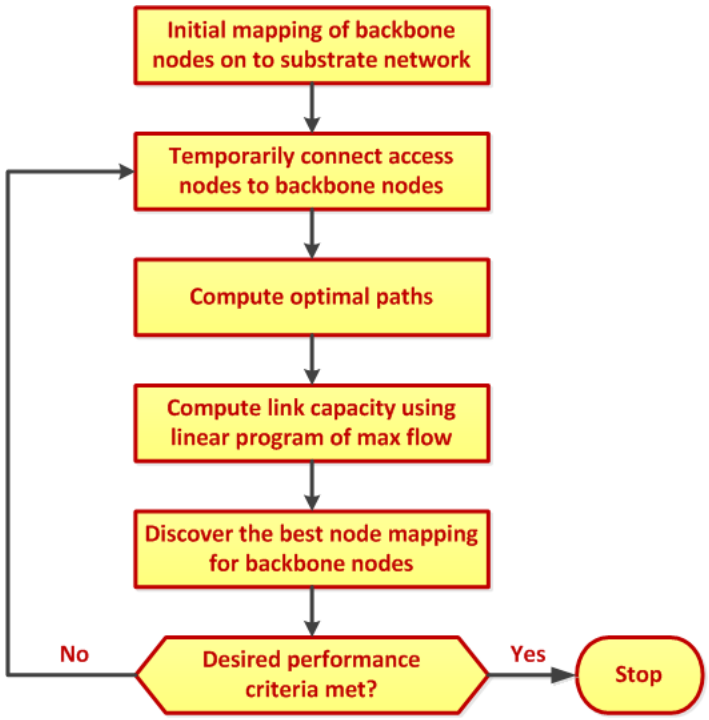

Figure 1.5: Backbone-star based static virtual network resource allocation

higher revenue and lower bandwidth cost than the baseline approach.

The next static virtual network allocation method has been shown in [22]. In their algorithm, Chowdhury et al. have proposed a method to map the virtual network onto the substrate network in the coordinated way. They have built the augmented graph based on the substrate graph and virtual network. Then they have proposed two methods - deterministic rounding virtual network embedding (DViNE) and randomized rounding based virtual network embedding (RViNE) - to search for the substrate nodes for hosting the virtual network demands. The DViNE algorithm searches substrate nodes satisfying location constraints with maximum availability to allocate to virtual network demands. RViNE also searches the nodes satisfying both location constraints and availability constraints, but the probability to allocate a substrate node to a virtual network demand is proportional with the substrate nodes' availability. After finding all substrate nodes to allocate for the virtual network demand, the algorithm uses multi-commodity flow [23] to allocate the resource for virtual links. The results show the advantages of these two algorithms in acceptance ratio, revenue and provisioning code; comparing with the previous methods: Greedy node mapping with shortest path based link mapping (G-SP), greedy node mapping with splittable link mapping using multi-commodity flow (G-MCF), deterministic node mapping with shortest path

based link mapping (D-ViNE-SP), deterministic node mapping with splittable link mapping (DViNE-LB).

The other method for allocating resources to virtual network demands across multi-domains is given in [24]. A distributed protocol name PolyViNE (policy-based virtual network embedding) has been designed for SPs to communicate with InPs as well as InPs to communicate with each other to provide resources to SPs in the way of upholding InP's internal policy. Basically, the protocol works as follows. When a SP sends an EMBED message to InPs to ask for embedding its virtual network demands, each InP checks whether it has enough resources to host the demand. If yes, then it sends the SUCCESS message back to the SP. If it can partly serve the virtual network demand, then it sends the message FORWARD to other InP to ask for hosting the rest of the virtual network demand. If the InP cannot host the virtual network demand at all, then it will send the message RELAY to the next InP for asking the next InP to host that demand. The procedure continues until the last InP or SP can receive the SUCCESS message.

Another method, trying to solve the problem of virtual network embedding across multiple substrates, is presented in [25]. In their paper, Houidi et al. have formulated an optimization problem for SPs to decide how they can best split their virtual network demand through many substrate networks. The optimization problem is a quadratic integer programming with the objective of minimizing the hosting cost. Unfortunately, this optimization problem is proven NP-hard, and the authors have proposed a heuristic method using max-flow/min-cut to solve the problem. Simulation results show that this proposed method exhibits reduced costs compared to existing ones.

In [26], an optimization problem is formulated to solve the mapping of virtual network to the substrate network. The problem tries to minimize the mapping cost for each virtual network request with the following constraints. Firstly, two virtual nodes from the same virtual network demand cannot be assigned to the same substrate node. Secondly, each virtual node cannot be assigned to more than one substrate node. Thirdly, a virtual node can only be mapped to the substrate node with enough resources. Finally, a virtual access node can be mapped to the substrate access node only. This integer programming problem is also proven NP-hard. The authors have used the ant-colony heuristic method (VNE-AC) to solve this optimization problem. The simulation results show that the proposed method makes embedding more effective, and achieves better resource utilization than the previous

methods such as VNE-Greedy, VNE-Cluster, VNE-Subdividing and VNE-Least.

In spite of many proposals for static virtual network allocation, they only concern how to provide an exclusive service to virtual network demands. However, in the environment of multi-tenant data center network, exclusive resource allocation for two tenants Google and Nasdaq as in Figure 1.1 is inefficient and uninterest in an economic viewpoint. We proposed an optimization framework using careful overbooking concept to provide additional services to exclusive one. The detailed formulation and experiment results will be presented in **Chapter** III.

1.2.3 Dynamic virtual network allocation

The dynamic virtual network allocation which periodically re-adjusts allocated virtual link bandwidth is proposed by [17]. In their paper, Zhu and Ammar have developed a selective virtual network reconfiguration scheme. They argue that the virtual network reconfiguration is more expensive than the flow re-routing problem [27] in both reconfiguration and service disruption cost. Furthermore, they show that the virtual network reconfiguration order affects to the substrate network performance. Therefore, they have developed a marking algorithm to mark the virtual networks' priorities for reconfiguring. This algorithm marks the high-stress nodes or links and reconfigures only the virtual networks hosting on that nodes or links accordingly. The simulation results show that the virtual network allocation with reconfiguration has lower maximum node and link stress. This method needs extensive computing resources because it considers the marked virtual networks as the new demands to be served.

Another method to deal with the dynamic virtual network allocation is Davinci [28]. In this paper, He et al. have presented an architecture, in which each substrate link periodically reassigns the bandwidth shares among its virtual links; while at a smaller time-scale, each virtual network independently runs a distributed protocol to maximize its utility objective. The resource allocation problems in both substrate network and virtual networks are derived by optimization theory [29]. The problems in virtual networks are formulated as network utility maximization [30] with the objective as throughput, delay, blocking probability and the constraints as virtual link's residual bandwidth. The main optimization problem in the substrate network is derived from the convex optimization theory with the objective function as the weighted sum of all virtual networks' utilities and the constraints as the substrate link's

bandwidth. Because of the computational complexity, a centralized algorithm for solving the main optimization problem is impossible to implement in a large network model such as the internet. The work has used primal decomposition [31] to decompose this main problem to many sub-problems each of which can be solved at a local node with the local information. The results of the individual sub-problems construct the solution for the main problem. The experiments show that the bandwidth shares not only converge quickly for a range of constant step size but also adapt swiftly to traffic shifts and link failures.

Zhou et al. [32] point out the weakness of [28]. If any of the virtual networks exhibit malicious behaviours, then unfair allocation will happen. To fix this weakness, they have used game theory to formulate a new virtual link bandwidth allocation problem. Their proposed game problem uses the economic incentives to prevent the virtual networks from exhibiting the malicious behaviours. In particular, they have added a new term, cost of hosting on a congested link, to their problem's pay-off function so that if a virtual network stresses the clogged physical links, then it will be charged with the high cost. They have proven that their game converges to Nash equilibrium, and proposed an algorithm to solve the problem iteratively. The simulation results show that their algorithm can rapidly achieve Nash equilibrium.

In [33], Wang et al. have modelled the virtual link allocation in network virtualization as a Stackelberg game. At the upper level, the virtual networks play a non-cooperative bandwidth allocation game. At the lower level, the substrate network intelligently sets a price for driving virtual networks to maximize the total revenue. After formulating the game problem, the authors have also proposed a distributed method of decomposing the global optimization problem into local algorithms. These algorithms can run independently at different ingress nodes of substrate network.

Davinci [28] does not allow a virtual link to traverse more than one physical link. It also does not permit a virtual link to host on another virtual link. In [15], Drwal and Gasior have reformulated the optimization problem to relax these above constraints. The solution for their optimization problem is a distributed virtual bandwidth allocation algorithm which grants the virtual links for the arbitrary traveling. Their algorithm's convergence is theoretically proven.

However, the above dynamic virtual network allocations only pay attention to the internet which is the large scale in both substrate network and virtual network. The only existing commercial network virtualization model now is a cloud resident data center which has dif-

ferent structure with the internet. The cloud resident data center has big size substrate, but small size virtual network. We have re-derived the dynamic virtual network allocation algorithm to work in cloud resident data center model by designing the centralized algorithm for virtual network and distributed algorithm for substrate network.

1.3 Objective of dissertation

The objective of this dissertation is to propose the algorithms of optimally allocating the data center resources to tenants in the context of network virtualization. As shown in Figure 1.1, the traffic from tenants of server hosting services are quite static. This works used static resource allocation approach and applied the careful overbooking concept to give more flexibilities in allocating virtual link bandwidth than the conventional exclusive allocation. However, the traffic in tenants with IaaS service is quite dynamic. This work adopts the new model of cloud resident data center to provide IaaS to tenants. Based on this model, this work has applied optimization theory to derive a new dynamic algorithm, which control traffic flowing in each virtual data center centrally and control traffic flowing in substrate network distributedly. The algorithm has been implemented in our small cloud resident data center testbed to show the better performance than the previous algorithms, i.e. [28].

1.4 Scope of dissertation

The scope of this dissertation in controlling the allocation of substrate resources to virtual networks is described as follows:

1. Study the dynamics of traffic in network virtualization
2. Apply a careful overbooking concept to derive the new static allocating method, which uses substrate capacity efficiently.
3. Study the theory for integrating the derived method of resource allocation to the optimization framework.
4. Write a simulation in MATLAB to quantify the results.
5. Study the practical network virtualization via cloud resident data center model

6. Build a small-scale test-bed of cloud resident data center based on the OpenFlow [12] framework.

7. Use optimization theory to derive the dynamic traffic control model on both virtual networks and substrate network.

8. Test the algorithm on our small-scale test-bed

1.5 Organization of dissertation

After the introduction in this chapter, the theoretical backgrounds and cloud resident data center testbed are presented in **Chapter** II. The purpose here is to provide the basic information of primal decomposition method, gradient projection method, congestion price and Karush-Kuhn-Tucker conditions; as well as to provide the practical settings of our small test-bed to implement and test our derived traffic control algorithms. The static optimal virtual network allocation with careful overbooking is explored in **Chapter** III to aggregate the server hosting services in multi-tenant data center network. Here, the careful overbooking concept is analyzed to calculate SLA parameters for guaranteeing the quality of service. A Mixed integer programming is formed by using the above SLA parameters to search for the best operational point of the overall network. In **Chapter** IV, an optimization problem is formulated from the new structure of cloud resident data center for IaaS service. The primal decomposition is used to decompose this optimization problem into a novel algorithm for dynamically adaptive traffic control. This algorithm is theoretically proven of convergence and implemented in simulation (MATLAB) as well as in testbed (OpenFlow system) to qualify the performance. This dissertation concludes all the contributions in **Chapter** V, together with possible future research directions. The **Appendix** gives detailed explanations to the centralized virtual network allocation algorithm.

CHAPTER II

THEORETICAL BACKGROUND AND CLOUD RESIDENT DATA CENTER TESTBED

This chapter describes the theoretical background and our small cloud resident data center testbed. The backgrounds of using optimization theory to model and solve the network virtualization problems are addressed in Section 2.1. Four issues are of particularly interest, namely, primal decomposition method, gradient projection method, congestion price, and Karush-Kuhn-Tucker (KKT) conditions. Section 2.2 then presents our small cloud resident data center testbed which is the only commercial network virtualization model now a day. The purpose here is to introduce our system model and the scheme to control traffic rate in each virtual network as well as the methods to control the traffic in the substrate network. The concluding remarks, which locate the applications of this chapter's basic knowledge in this dissertation, are presented in Section 2.3.

2.1 Theoretical background

2.1.1 Primal decomposition

In such a large Amazon cloud network [34], which connects many data centers throughout USA, EUROPE and ASIA; no single server is strong enough to solve the centralized algorithm for optimal rates of all network's nodes. Hence, the centralized optimization algorithm should be replaced by the distributed one. The distributed optimization algorithm decomposes the centralized optimization problem to be many sub-problems each of which can be solved independently in the equivalent network's node. A simple example of a decomposable optimization problem is:

$$\begin{aligned} \underset{\{x_1,x_2\}}{\text{maximize}} \quad & x_1 + x_2 \\ \text{subject to} \quad & x_1 \leq 1 \\ & x_2 \leq 1 \end{aligned} \tag{2.1}$$

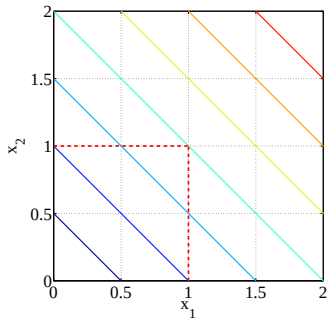

Figure 2.1: Decomposable problem

where x_1, x_2 are the variables that we need to find in the constraint set $x_1 \leq 1; x_2 \leq 1$ to maximize the objective function of $x_1 + x_2$. This problem can be decomposed into two sub-problems each of which varies with either x_1 or x_2:

$$\begin{aligned} \underset{\{x_1\}}{\text{maximize}} \quad & x_1 \\ \text{subject to} \quad & x_1 \leq 1 \end{aligned} \tag{2.2}$$

$$\begin{aligned} \underset{\{x_2\}}{\text{maximize}} \quad & x_2 \\ \text{subject to} \quad & x_2 \leq 1 \end{aligned} \tag{2.3}$$

As in Figure 2.1, the combination of the sub-problem's results, (x_1^*, x_2^*), is the optimal point for the original optimization problem.

However, not all the optimization problems are decomposable. For instance, let examine the following simple optimization problem:

$$\begin{aligned} \underset{y,\{x_1,x_2\}}{\text{maximize}} \quad & x_1 + x_2 \\ \text{subject to} \quad & x_1^2 + (1-y)^2 \leq 1 \\ & x_2^2 + y^2 \leq 1 \end{aligned} \tag{2.4}$$

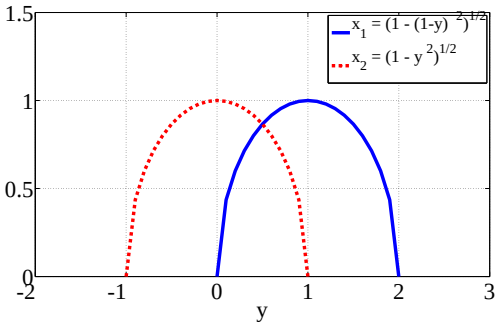

Figure 2.2: The two optimal values of y

If we decompose this optimization problem into two sub-problems as follows:

$$\begin{aligned} \underset{\{y,x_1\}}{\text{maximize}} \quad & x_1 \\ \text{subject to} \quad & x_1^2 + (1-y)^2 \leq 1 \end{aligned} \tag{2.5}$$

$$\begin{aligned} \underset{\{y,x_2\}}{\text{maximize}} \quad & x_2 \\ \text{subject to} \quad & x_2^2 + y^2 \leq 1 \end{aligned} \tag{2.6}$$

The optimal point of problem (2.5) is $x_1^* = 1$ when $y_1^* = 1$, and the optimal point of problem (2.6) is $x_2^* = 1$ when $y_2^* = 0$. Because y_1^* in (2.5) is different from y_2^* in (2.6), we cannot combine the optimal points from the sub-problems into the optimal point of the original problem. The coupling variable y makes this problem undecomposable. To convert an undecomposable optimization problem into decomposable one, the work in [31] has surveyed many methods such as dual decomposition for coupling constraint problem, primal decomposition for coupling variable problem, and the combination of the two decomposition methods: partial-dual, primal-dual, dual-primal.

The centralized optimization proposed in **Chapter** IV has coupling constraints, hence we apply the primal decomposition method to decompose the problem. In general, the primal decomposition method can be explained for the coupling-constraint convex optimization problem follows.

$$\begin{aligned} \underset{y,\{x_i\}}{\text{maximize}} \quad & \sum_i f_i(x_i) \\ \text{subject to} \quad & x_i \in \mathcal{X}_i \quad \forall i \\ & A_i x_i \leq y \ \forall i \\ & y \in \mathcal{Y} \end{aligned} \tag{2.7}$$

where $f_i(x_i)$ is convex function with x_i, and does not vary with x_j if $i \neq j$. The notation $\{x_i\}$ presents the set of all x_i. This problem is undecomposable because of the coupling variable y in the constraint $A_i x_i \leq y$. The primal decomposition method makes this problem decomposable by first fixing y as any point in the feasible set $y \in \mathcal{Y}$ to find $x_i^* \ \forall i$.

$$\begin{aligned} \underset{\{x_i\}}{\text{maximize}} \quad & \sum_i f_i(x_i) \\ \text{subject to} \quad & x_i \in \mathcal{X}_i \quad \forall i \\ & A_i x_i \leq y \ \forall i \end{aligned} \tag{2.8}$$

After fixing y, the original problem (2.7) only has variables of $\{x_i\}$ as in (2.8), and can be decomposed into many sub-problems each of which with an index number i is presented as

follows:

$$\begin{aligned} \underset{x_i}{\text{maximize}} \quad & f_i(x_i) \\ \text{subject to} \quad & x_i \in \mathcal{X}_i \\ & A_i x_i \leq y \end{aligned} \tag{2.9}$$

Each sub-problem with index number i can be solved locally in local contraint set X_i and $A_i x_i \leq y$ to find the optimal point x_i^*. After getting x_i^* with $\forall i$, we find the new optimal $y \in \mathcal{Y}$ by forming the following master problem:

$$\begin{aligned} \underset{y}{\text{maximize}} \quad & \sum_i f_i^*(y) \\ \text{subject to} \quad & y \in \mathcal{Y} \end{aligned}$$

where $f_i^*(y) = f_i(x_i^*, y)$ is the function of variable y when replacing x_i by the optimal point x_i^* from the sub-problems

For example, we can apply primal decomposition method to decompose the problem (2.4) by first fixing the coupling variable y. The sequel problem is as follows.

$$\begin{aligned} \underset{\{x_1,x_2\}}{\text{maximize}} \quad & x_1 + x_2 \\ \text{subject to} \quad & x_1^2 + (1-y)^2 \leq 1 \\ & x_2^2 + y^2 \leq 1 \end{aligned} \tag{2.10}$$

This problem is decomposable into the two following sub-problems:

$$\begin{aligned} \underset{\{x_1\}}{\text{maximize}} \quad & x_1 \\ \text{subject to} \quad & x_1^2 + (1-y)^2 \leq 1 \end{aligned} \tag{2.11}$$

$$\begin{aligned} \underset{\{x_2\}}{\text{maximize}} \quad & x_2 \\ \text{subject to} \quad & x_2^2 + y^2 \leq 1 \end{aligned} \tag{2.12}$$

The optimal point for (2.11) is $x_1^* = \sqrt{1-(1-y)^2}$, and the optimal point for (2.12) is $x_2^* = \sqrt{1-y^2}$. Replace these optimal points back to the original optimization problem to have the new optimization problem with only variable y.

$$\underset{y}{\text{maximize}} \quad \sqrt{1-(1-y)^2} + \sqrt{1-y^2}$$

And the optimal solution is 0.37 when $y^* = 2.1140$

2.1.2 Gradient projection method

The analysis in **Chapter** IV for the master problem (4.3) needs to solve a relaxed optimization problem with simple non-negative orthant constraints as follows.

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & f(x) \\ \text{subject to} \quad & x \geq 0 \end{aligned} \tag{2.13}$$

In the optimization problem, $f(x)$ is the objective function, $x \in R^n$ is the independent variable vector and $x \geq 0$ is the non-negative orthant constraint.

An iterative method is a general solution for the simple-constrained optimization problem (2.13). In each iteration, the method relaxes the non-negative orthant constraint to solve the sequel unconstrained problem $\underset{x}{\text{minimize}} f(x)$, then projects the intermediate results back to the non-negative orthant constraint set $x \geq 0$ to make them feasible. For solving the relaxed unconstrained problem, two main methods, namely, gradient ascent method and Newton's method, have been presented in [35]. Because the Newton's method needs to calculate the second-order derivative of the optimization's objective function, it increases the computational complexity significantly. This dissertation decides to use the second one, gradient ascent method, which requires only first-order derivative of the optimization's objective function $f(x)$. Furthermore, this first-order derivative is exactly the congestion price vector which can be taken from previous dual solutions of optimization sub-problems (4.1).

A combination of the gradient ascent method and projection method is used in this dissertation under the name of gradient projection method. In each iteration of the gradient projection method, the gradient ascent algorithm is used to solve the relaxed unconstrained problem. The projection method is used next for projecting the intermediate results to the feasible set.

The searching vector $s(x) \in R^n$ of the gradient ascent method is set to be the gradient of the objective function $f(x)$.

$$s(x) = \nabla f(x)$$

It can be observed that in the unit norm constraint, the searching vector $s(x)$ points to the highest-increased direction of the objective function $f(x)$. The observation can be proven by using the first-order Taylor approximation to approximate the objective function $f(x)$ at the

point $x + s(x)$.

$$f(x + s(x)) \approx f(x) + \nabla^T f(x)s(x) \tag{2.14}$$

From (2.14), the unit-norm searching vector $s(x)$ to maximize $f(x + s(x))$ is equal to the unit-norm searching vector $s(x)$ to maximize $\nabla^T f(x)s(x)$. Applying the Cauchy-Schwarz inequality:

$$\begin{aligned} \nabla^T f(x)s(x) &\leq ||\nabla f(x)||_2 ||s(x)||_2 \\ &= ||\nabla f(x)||_2 \\ &= \nabla^T f(x)\nabla f(x) \end{aligned} \tag{2.15}$$

Hence, the unit-norm $s(x) = \nabla f(x)$ will maximize $f(x + s(x))$ with regard to $||s(x)|| = 1$. That is, at every iteration, the searching vector in gradient ascent method points to the highest-increased direction of the objective function.

The justification for the projection method, which is used in each iteration of gradient ascent method, to find the feasible optimal solution of (2.13) is presented in Figure 2.3. The figure only explains the one-dimensional optimization problems, but it is always possible to extend the explanation for higher-dimensional optimization problems. Figure 2.3 shows the objective function $f(x)$ which has an infeasible maximum (**A**) at the equivalent infeasible optimal point (**C**). Because of the concavity of the objective function $f(x)$, it is always possible to project the infeasible optimal point (**C**) to the feasible set, which in here is the non-negative quadrant, to get the feasible optimal point (**O**). The solution is (**B**), which is the maximum of the objective function $f(x)$ in the feasible set of non-negative quadrant.

The gradient projection algorithm which applies the gradient ascent and projection method to find the feasible optimal solution for simple-constrained optimization problem is presented as follows.

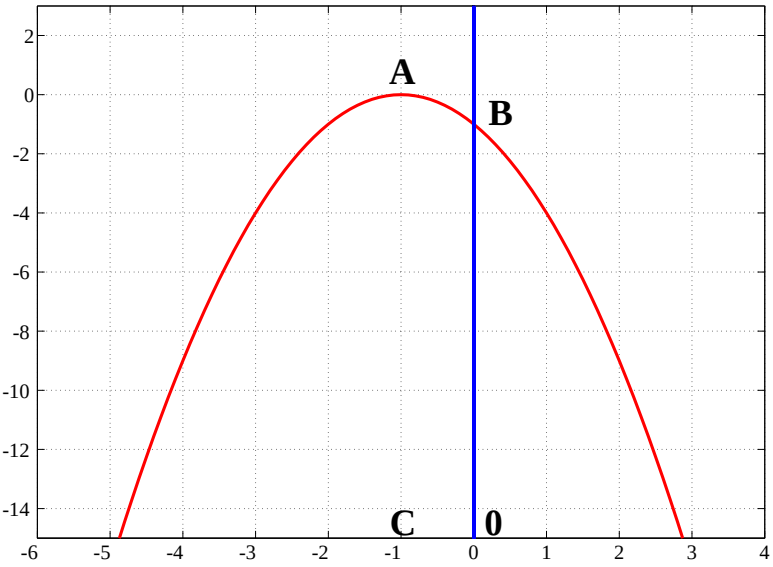

Figure 2.3: Projection method

Algorithm II.1 Gradient ascent method

Suppose that a function $f : \mathbb{R}^n \to \mathbb{R}$ and a starting point $x_0 \in \mathbb{R}^n$ are given. Let $k = 0$.

1: **while** Termination criterion is not satisfied **do**
2: Set $k = k + 1$
3: Set $s_k = \nabla f(x_k)$
4: Find β_k to minimize $f(x_k + \beta_k s_k)$ with respect to β_k
5: Set $x_{k+1} = [x_k + \beta_k s_k]_p$
6: **end while**

where $[\bullet]_p$ is the projection to feasible constraints; k is an iteration index number; β is step size, which can be found by back tracking line search [29].

2.1.3 Congestion price

The argument that the congestion price is the partial derivative of the primal objective function with right hand side of inequality constraint is used in (4.6). This section proves the above argument in a convex optimization problem with affine inequality constraints [29].

$$\begin{aligned} \underset{x}{\text{maximize}} \quad & f(x) \\ \text{subject to} \quad & Ax \leq b \end{aligned} \tag{2.16}$$

where $x \in R^n$, $A \in R^{m \times n}$ and $b \in R^m$. The perturbed problem is derived from the primal one (2.16) by relaxing the inequality constraints by u units.

$$\begin{aligned} \underset{x}{\text{maximize}} \quad & f(x) \\ \text{subject to} \quad & Ax \leq b + u \end{aligned} \tag{2.17}$$

If the primal problem (2.16) has the optimal solution $p^*(0)$, then the perturbed problem has the optimal solution $p^*(u)$. The dual of the perturbed problem (2.17) is derived by first calculating its Lagrange function.

$$\begin{aligned} \mathcal{L}(x, \lambda) \quad &= f(x) + \lambda^T (b + u - Ax) \\ \nabla_x \mathcal{L}(x, \lambda) &= \nabla_x f(x) - A^T \lambda \end{aligned} \tag{2.18}$$

where $\lambda \in R^m$ is Lagrange multiplier vector. Then, the Lagrange function $\mathcal{L}(x, \lambda)$ is maximized with the primal variable x by setting $\nabla_x \mathcal{L}(x, \lambda) = 0$. Hence, the dual of the perturbed problem (2.17) is found by minimizing the resulted Lagrange function $\max_x \mathcal{L}(x, \lambda)$ with dual variable λ.

$$\begin{aligned} \underset{\lambda}{\text{minimize}} \quad & f(x) + \lambda^T (b + u - Ax) \\ \text{subject to} \quad & \nabla_x f(x) - A^T \lambda = 0 \end{aligned}$$

Because the strong duality holds for the perturbed problem, the primal solution $p^*(u)$ is equal to the dual solution $d^*(u)$.

$$p^*(u) = d^*(u) \tag{2.19}$$

$$\begin{aligned} \frac{\partial}{\partial u_i} p^*(u) &= \frac{\partial}{\partial u_i} d^*(u) \\ &= \frac{\partial}{\partial u_i} \left(f(x) + \lambda^{*T} (b + u - Ax) \right) \\ &= \lambda_i^* \end{aligned} \tag{2.20}$$

The result in (2.20) proves that the congestion price λ_i^* is the partial derivative of primal objective function of (2.16) with the right hand side in equality contraint u_i.

2.1.4 Karush Kuhn Tucker conditions

This section describes Karush Kuhn Tucker (KKT) conditions to prove the optimality of a general convex optimization problem.

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & f(x) \\ \text{subject to} \quad & f_i(x) \leq 0, \quad i = 1, .., m \\ & h_j(x) = 0, \quad j = 1, .., p \end{aligned} \tag{2.21}$$

where $f(x)$ is convex objective function; $f_i(x)$ is affine $\forall i$, and $h_j(x)$ is affine $\forall j$. Let $\tilde{x}$ be the feasible points of (2.21); and $\tilde{\lambda}, \tilde{\nu}$ be the feasible points of the dual problem of (2.21). If $\tilde{x}, \tilde{\lambda}, \tilde{\nu}$ are satisfied by KKT conditions, then $\tilde{x}$ will be primal optimal point, and $\tilde{\lambda}, \tilde{\nu}$ are dual optimal points of the convex optimization problem (2.21) [29].

1. Primal constraints: $f_i(x) \leq 0, \quad i = 1, .., m$ and $h_j(x) = 0, \quad j = 1, .., p$
2. Dual constraints: $\lambda \geq 0$
3. Complementary slackness: $\lambda_i f_i(x) = 0 \quad , i = 1, .., m$
4. Gradient of Lagrange with respect to x vanishes:

$$\nabla f(x) + \sum_{i=1}^{m} \lambda_i \nabla f_i(x) + \sum_{i=1}^{p} \nu_i \nabla h_i(x) = 0 \tag{2.22}$$

2.2 Cloud resident data center testbed

Figure 2.4: Physical model

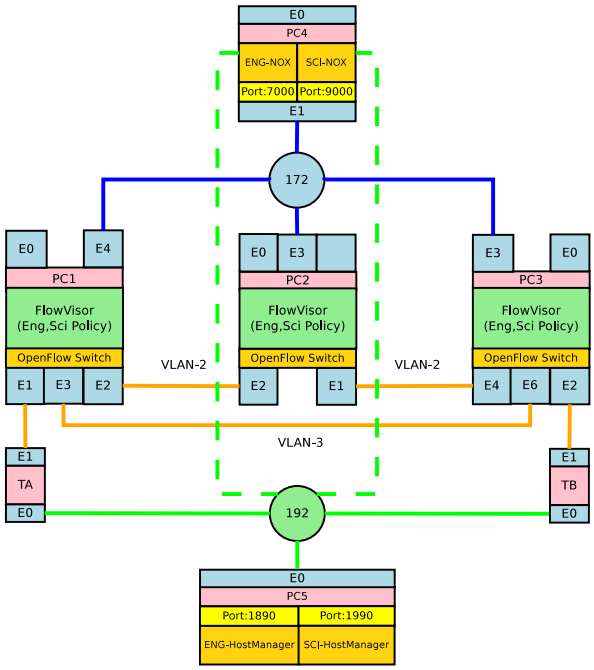

Figure 2.5: Logical model

A theoretically derived algorithm is not convinced enough if it has not been evaluated and continuously improved in a realistic environment which is a global cloud network in our research. In spite of the importance, the integration of our theoretically derived algorithm into such a large network is extremely challenging. To cope with this challenge, we decide to build a small PC based testbed for implementing our algorithm as first step to realize the practical problems of implementation. Our OpenFlow based cloud resident data center testbed is presented in Figure 2.4 with 7 PCs. PC1-3 are installed with OpenVswitch; PC4 is installed with NOX; PC5 is installed with host manager; PC6-7 are end hosts. The software modules in each PC such as OpenVswitch, NOX are the control modules used in the current cloud network.

Our logical model of the testbed is shown in Figure. 2.5. The two NOX controllers, ENG and SCI, are installed as computer processes in PC4 with the communicating port 7000 for ENG-NOX and 9000 for SCI-NOX. The two host managers, ENG and SCI, are also installed as computer processes in PC5 with the communicating port 1890 and 1990

accordingly. The control network 172 is used for NOX to control the OpenVswitches. The managed network 192 is used to remotely access to all the devices in the network and for the host managers to communicate with end hosts. The data network is used for the switches to transmit the traffic from one end host to others.

In the following, we explain our mechanisms to control the traffic Controller, in each virtual network as well as in the substrate network.

2.2.1 Traffic control in each virtual network

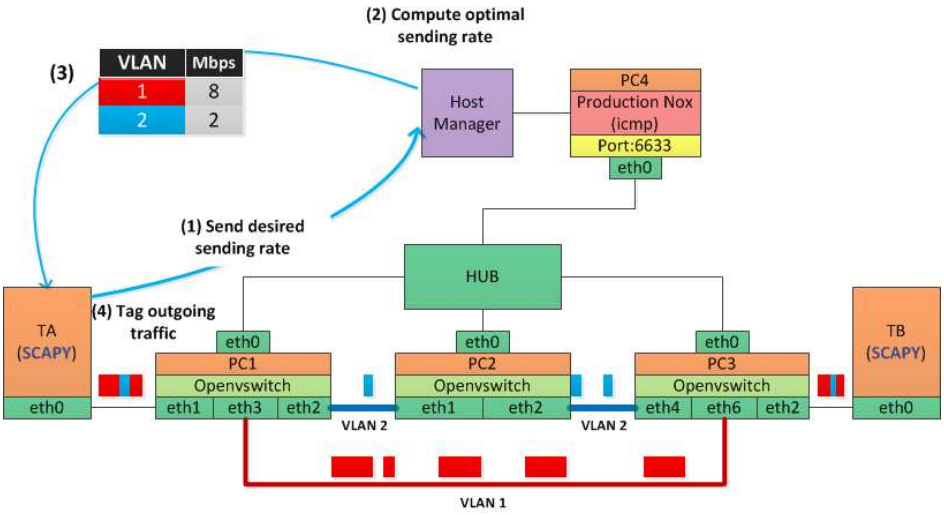

Figure 2.6: Virtual network control

The traffic control in each virtual network is the basic step for the whole cloud network traffic control. The purpose of virtual-network traffic control is to build the mechanisms and protocols for the host manager to communicate and control the sending traffic of the all virtual network's end hosts. From Figure 2.6, to practicalize the purpose, we need to deal with the problems in controller, host manager as well as in end hosts. The problems from controller are to differentiate the virtual-network's source-destination paths, and forward the packet streams through those paths. The problems from host manager are to synchronizingly communicate with all end hosts to get the traffic request as well as update the new optimal traffic rate, and to calculate the optimal rate for each end host. The problems from end hosts are to build the packets, and to send the packet out each path with the allocated rate from host manager, and to communicate with the host manager for the requested rate. The solutions for problems in controller will be presented in the controller implementation. The solution for problems in host manager will be presented in host manager implementation. The solution

for the problems in end host will be presented in end host implementation. Inspired from the general idea of [36], we have scratched the overall process to synthesize all the mechanisms in controller, in host manager and in end host to control the traffic flowing in each virtual network.

1. When receiving the requests from host manager, all the end hosts will immediately send their desired sending rate request (1).

2. Host manager calculates the optimal traffic rates based on all end hosts' desired sending rates and the network state at that time (2).

3. Host manager sends back the optimal rate in the form of (VLAN, Mbps) to each end host (3).

4. Each end host sends the traffic to the paths based on the allocated path rate (4).

NOX controller implementation

In a virtual network, for an end host to send traffic via multipath, NOX controller needs to teach OpenVswitches to differentiate the virtual-network source-destination paths, and to forward the packet stream through these paths. We use VLAN for NOX to differentiate the paths in our small system. For example, there are two paths from end host TA to end host TB. The upper path is tagged with VLAN ID = 2, and the lower path is tagged with VLAN ID = 3. In case of the big system with so many paths that VLAN does not have enough address to tag, we can use MPLS to tag the paths instead. The forwarding entries in the OpenVswitches of our system can be simplified to show in Figure 2.7.

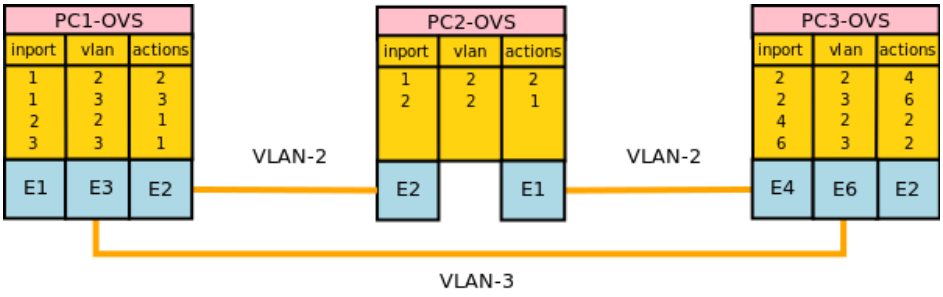

Figure 2.7: OpenVswitch forwarding entries

where *inport* is physical port number of OpenVswitch, *actions* has the output port to which the switch should forward the packet. *inport* and *vlan* of incoming packet are the conditions

for the switch to do the *actions*. From the figure, packets with VLAN ID = 2 will be forwarded to the upper path, and packets with VLAN ID = 3 will be forwarded to the lower path.

We have written a python script for NOX controller to teach OpenVswitch to forward the packet stream based on VLAN information as follows:

Algorithm II.2 Traffic control script in NOX controller

```
1: def datapath_join_callback()
2: def handle_port_status()
3: install(self):
4:     self.register_for_datapath_join()
5:     self.register_for_port_status()
```

When NOX start, it will run this script which does the followings. The script registers with NOX controller that it will process the datapath join event which is generated if a new switch connects to the NOX controller (line 4). This event is processed by the actions which are defined in the function of line 1. In our system, we control NOX to add the forwarding entries to OpenVswitches whenever a datapath join event comes to the NOX. The function in line 5 is used to register with NOX controller that this script will process the event of port status. And we define our actions in the equivalent function in line 2. This is used to test the case of failure recovery in **Chapter** IV.

Host manager implementation

A virtual-network's host manager is the center to calculate the traffic rate for all end hosts in the virtual network. The host manager can be integrated in the virtual network's controller or be an independent module. The host manager connects to all end hosts of a virtual network via a socket. For example, the socket of host manager for virtual network ENG is ('192.168.0.5',1890), and for virtual network SCI is ('192.168.0.5', 1990). The host manager synchronizes the traffic rate of all virtual-network's end hosts by periodically sending the requests and updating the new rate for end hosts. After getting the traffic requests from all end hosts, the host manager calculates the optimal rate for all end hosts by solving the optimization in **Chapter** IV using CVXOPT package [37].

We have written a socket program in Python for the host manager to connect with all

end hosts in the virtual network as follows:

Algorithm II.3 Class Vncal: Traffic control in a virtual network

1: INIT: $T = 5$ seconds
2: class Vncal(Thread):
3: connect_to_endhosts
4: **while** $True$ **do**
5: **for all** endhosts **do**
6: $r_i^{(k)}$ = get_max_rate(end-host_id)
7: **end for**
8: $\vec{z}^{(k)}, \vec{\lambda}^{(k)}$ = nw_opt_rate(max_rate_list)
9: **for all** end-hosts **do**
10: sendrate($\vec{z}^{i(k)}$, end-host_id)
11: update2hm($\vec{\lambda}^{(k)}$)
12: **end for**
13: **end while**

where the function in line 3 creates a socket ('IP', PORT) to connect with all the virtual-network's end hosts. For example, in virtual network ENG, the socket for host manager to connect to end host TA is ('192.168.0.14', 35619), to end host TB is ('192.168.0.15', 58468). The function in line 6 asks the end host with its *end_host_id* for the request rate and put them to a list *max_rate_list*. The function in line 8 solves the network optimization which is presented in **Chapter** IV for the optimal rate using the Netcal class which will be presented next. The returned values of function in line 8 are the optimal rate $z^{i(k)}$, which will be used in function in line 10 to update back to the end host, and congestion price $\vec{\lambda}^{(k)}$, which will be used in the function in line 11 to update the new optimal rate back to the end host.

We also have written a program for the host manager to calculate the optimal rate of all virtual-network's end hosts using CVXOPT [37]. Because the optimization in each virtual network is a convex problem, the primal-dual interior point algorithm has been chosen to solve the problem centrally. The algorithm is encoded in the Vncal class which is outlined as follows:

Algorithm II.4 Class Netcal: calculate the optimal rate and price, file: grad.py

1: **INIT:** $q, I, J, IJ = I * J, y^{(k)}$
2: class Netcal(object)
3: $rhs2st$ = matrix(numpy.zeros(self.IJ))
4: **for** l in L **do**
5: **for** i in I **do**
6: **for** j in J **do**
7: **GH**1$[l][i * J + j] =$ **H**$[i][l][j]$
8: **end for**
9: **end for**
10: **end for**
11: **GH** = matrix(**GH1**)
12: **DI** = -numpy.identity(IJ)
13: **G** = matrix(numpy.vstack((**GH**, **DI**)))
14: $\vec{h}$ = matrix(numpy.vstack(($y^{(k)}$, $\vec{rhs2st}$)))
15: sol = solvers.cp(F, **G**, $\vec{h}$)

where I is the number of traffic demands for each end hosts, J is the number of paths for each demand, L is the number of links in the network. Basically, the program calculates the parameters $F, \mathbf{G}, \vec{h}$ for the optimization solver function in line 16. The function F(x,z) return value from its input parameters x, z for the function in line 16 to solve the optimization problem numerically. Matrix **G** is the gradient matrix of the objective function and $\vec{h}$ is the right-hand side of the constraint set.

End host implementation

End host in our testbed is a virtual machine hosting on a physical machine. Every end host has to communicate the optimal rate with host manager, and build packets as well as send these packets to destination through multipath with pre-determined optimal rate.

For the first mission, the end hosts need to figure out the wanted rate to send to each destination at the beginning of each period T. The end hosts in our system figure out the wanted rate by periodically monitoring the number of buffered packets in the output port. Based on this information, the end host predicts its desired sending rate of next period. This

Listing II.1: Packet header

Frame 374899: 1500 bytes on wire (12000 bits), 1500 bytes captured (bits);
Ethernet II (VLAN tagged), Src: 08:00:27:d8:ce:37,Dst: 08:00:27:c1:c5:8a;
Internet Protocol Version 4, Src: 127.0.0.1 (127.0.0.1),Dst:127.0.0.1(127.0.0.1);
Transmission Control Protocol, Src Port: 23335 (23335), Dst Port: 12336 (12336), Seq: 1

Frame 374899: 1500 bytes on wire (12000 bits), 1500 bytes captured (12000 bits);
Ethernet II (VLAN tagged), Src: 08:00:27:d8:ce:37,Dst: 08:00:27:c1:c5:8a;
Internet Protocol Version 4, Src: 127.0.0.1 (127.0.0.1),Dst: 127.0.0.1 (127.0.0.1);
User Datagram Protocol, Src Port: 23335 (23335), Dst Port: 12336 (12336), Seq: 1

desired sending rate then is sent as a request to host manager.

The second mission of the end host includes building packets and sending them to the destination via multipath with the pre-determined rate. We used Scapy [38] to build the wanted packet. For example, in our system, we need four kinds of packets:

- Packets for the switch of ENG slice to be forwarded via VLAN 3.
- Packets for the switch of ENG slice to be forwarded via VLAN 2.
- Packets for the switch from SCI slice to be forwarded via VLAN 3.
- Packets for the switch from SCI slice to be forwarded via VLAN 2.

All kinds of packets have the same size of 1500 bytes per packet. The four kinds of packets have the header as List II.1

We exploited tcpreplay [39] to send the traffic out from the end host. Tcpreplay can stimulate the real traffic by capturing the traffic to a .pcap file then use that .pcap file to send the traffic out. In our system, we build the desired packets using scapy [38] and send it to the destination. Tcpdump [40] is used to record the packet which can then be replayed with desired sending rate by tcpreplay. Each tcpreplay session is called in a process which kills itself after sending all the pre-determined amount of packets. To avoid the problem of the process built up in the system, we have written the program to send the desired amount of packets for each period T out as fast as possible. This kind of control is similar to TCP

control in the way of sending packet in a period of time, but different in the way of getting the rate information. In TCP, the end host figures out the amount of traffic to send from some indicators such as: blocking probability and delay. However, in our system, the end host receives the rate from the host manager. The algorithm in the end host is as follows:

Algorithm II.5 Virtual network ENG's end-host traffic control script in TA

1: INIT: MGR_IP_ADDRESS = $192.168.0.5$, MGR_PORT = 1890, $T = 5s$
2: p = packet_builder()
3: s = connect_to_hm()
4: **while** True **do**
5: wait_for_request_from_mgr()
6: send_max_rates()
7: wait_for_optimal_rates()
8: $n_1^{(1)} = \frac{z_1^{(1)} * T * 102400}{12000}$
9: $n_2^{(1)} = \frac{z_2^{(1)} * T\_s * 102400}{12000}$
10: t2 = Thread(target=path_traff_send, args=(2,vlan2_npackets,))
11: t2.start()
12: t3 = Thread(target=path_traff_send, args=(3,vlan3_npackets,))
13: t3.start()
14: **end while**
15: **Function** path_traff_send(vlan,npackets)
16: sendpfast(vlan=vlan,slice="eng",limit=npackets,iface="eth1")
17: **EndFunction**

where the function in line 2 uses Scapy [38] to build packets with the header of VLAN ID = 2 as well as VLAN ID = 3 to send through the two paths of our system; MGR_IP_ADDRESS is the ip address of host manager; MGR_PORT = 1890 is the TCP port number for the end-host to connect to host manager. The function in line 3 helps the end-host to initialize connection to host manager using the socket (MGR_IP_ADDRESS, MGR_PORT). The function in line 5 prepares end-host for receiving the request from host manager to send the traffic request by the function in line 6. The function in line 7 prepares end-host to wait for the optimal rate sent back from host manager. From the received optimal rate $z_1^{(1)}, z_2^{(1)}$ (100 Kbps), end-host calculates the number of packets with the size 1500 bytes to send out the path VLAN 2 ($n_1^{(1)}$) and VLAN 3 ($n_2^{(1)}$). The end-host sending the traffic creates two threads. The thread in line

10-11 sends the VLAN 2 packets to the output port eth1 and the thread in line 12-13 sends VLAN 3 packets to the output port eth1.

2.2.2 Traffic control in the substrate network

The virtual network's host manager can automatically update it virtual link bandwidth by implementing our substrate-network distributed algorithm. In each host manager, we have created three threads, which are ENG thread, SCI thread and substrate thread. In the ENG and SCI threads, we have implemented our virtual-network centralized algorithm, which is explained in **Chapter** IV for the host manager to control the traffic flowing inside its virtual network. In substrate thread, we have implemented substrate-network distributed algorithm to take the congestion prices from the previous thread, and calculate the new virtual link bandwidth for the virtual network, and update the new virtual link bandwidth back to the ENG and SCI thread. The algorithm in the substrate thread is sketched as follows:

Algorithm II.6 Calculate and update the virtual link bandwidth file:hm.py

1: **NOTE:** After each T, virtual link bandwidth is updated and the centralized optimization solver is run once
2: **INIT:** $T, c_l(0), y_l^{(1)}(0), y_l^{(2)}(0)$
3: eng = Vncal('eng', eng_Q, eng_cv, $y^{(1)}(t)$, T)
4: sci = Vncal('sci', sci_Q, sci_cv, $y^{(2)}(t)$, T)
5: **while** $True$ **do**
6: $\mu^{(1)}(t)$ = get_cp(eng_cv,eng_Q)
7: $\mu^{(2)}(t)$ = get_cp(sci_cv,sci_Q)
8: **for** $l\ in\ L$ **do**
9: $\tilde{y}_l^{(1)}(t+T) = y_l^{(1)}(t) + \frac{y_l^{(2)}(t)\left(\lambda_l^{(1)}(t)-\lambda_l^{(2)}(t)\right)}{y_l^{(1)}(t)+y_l^{(2)}(t)}$
10: $\tilde{y}_l^{(2)}(t+T) = y_l^{(2)}(t) + \frac{y_l^{(1)}(t)\left(\lambda_l^{(2)}(t)-\lambda_l^{(1)}(t)\right)}{y_l^{(1)}(t)+y_l^{(2)}(t)}$
11: // renormalize $\tilde{y}_l^{(k)}$
12: $y_l^{(1)}(t+T) = \frac{\tilde{y}_l^{(1)}(t+T)}{\tilde{y}_l^{(1)}(t+T)+\tilde{y}_l^{(2)}(t+T)} c_l$
13: $y_l^{(2)}(t+T) = \frac{\tilde{y}_l^{(2)}(t+T)}{\tilde{y}_l^{(1)}(t+T)+\tilde{y}_l^{(2)}(t+T)} c_l$
14: **end for**
15: eng.update($y_l^{(1)}(t+T)$)
16: sci.update($y_l^{(2)}(t+T)$)
17: **end while**

where ENG and SCI are the two virtual network threads. The function in line 7 gets the congestion price from ENG thread to the substrate thread. The function in line 8 gets the congestion price from SCI thread to the substrate thread. The function in line 16 updates the new virtual link bandwidth to ENG thread. The function in line 17 updates the new virtual link bandwidth to SCI thread.

2.3 Concluding remarks

In this chapter, the basic ideas of formulating and solving our optimization framework, and our small cloud resident data center testbed have been detailed. In particular, in Section 2.1, the primal decomposition method is used in Section 4.3.3 to decompose our cloud resident data center optimization problem; the gradient projection method is used in (4.5) to

iteratively find the solution for (4.4); congestion price is used in (4.6); KKT conditions are used in Section 4.3.4 to prove the convergence of our derived substrate-network distributed algorithm. The cloud resident data center testbed, which is presented in Section 2.2, is used in Section 4.4 to test our derived substrate-network distributed algorithm.

In the next chapter, the careful overbooking concept will be analyzed for providing SLA guarantee to network virtualization customers. We then formulate a Mixed Integer Programming problem using the analysis's results to control the traffic flowing in the network. The simulation results will be reported accordingly.

CHAPTER III

OPTIMAL STATIC VIRTUAL NETWORK ALLOCATION WITH CAREFUL OVERBOOKING

When deriving the virtual network embedding (VNE) algorithms, previous researches assume that an InP only provides the exclusive or best-effort services to embed virtual network demands in the InP's network [2]. In the exclusive service, a VNE algorithm either allocates 100% of what virtual networks require or blocks the virtual networks demanding more resources than what the substrate network can allocate [22, 41]. This "all or nothing" policy makes the InPs' resource allocation simple, but inflexible. In reality, existing virtual networks designed such as for email and ftp do not need the exclusive service. Virtual network managers are unhappy if being charged with the exclusive service without the necessity for such a high quality. In the best-effort service, a VNE algorithm does not take virtual network demands into account. It only tries to allocate network resources fairly among all VNs [17, 19]. But many virtual networks such as for voice, vdo, smartTV and IPTV need QoS guarantees. Even getting cheap price, virtual network managers are unhappy if the allocated quality is not good enough for them to operate their services.

As shown in Figure 3.1, to cope with the service providing problem, this chapter has adopted a careful overbooking method which let provider accept more reservations than the resources it actually has, but still can guarantee a level of quality (SLA) offered to virtual network subscribers. The virtual networks which do not need to have the exclusive service can be offered some other kinds of services whose quality can be specified by three parameters: probability of getting full availability: $1 - \delta$, probability of getting limited availability: $1 - \epsilon$, and reduction factor: γ. With these types of services, our system can figure out the actual resources for customers to guaranteeing the quality of service even when the system is in the most congested time. Furthermore, an optimization model [10] has been formulated in this chapter to route the traffic bandwidths that have been figured out from the previous step through the network. To provide flexible services, the aggregated demands which consist of many user requests have to be used. These kinds of demands are not suitable to

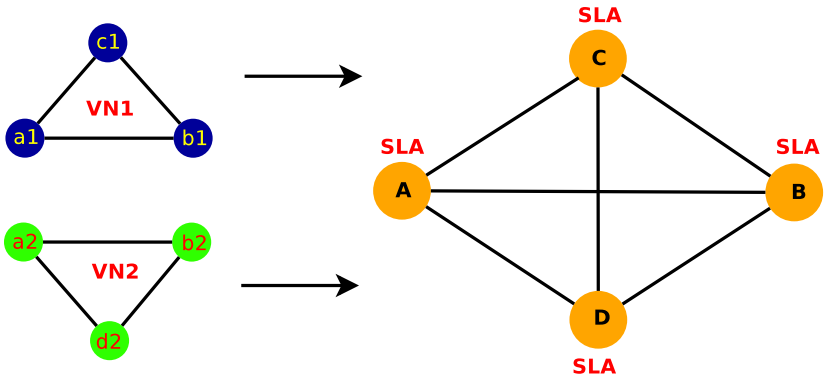

Figure 3.1: Virtual network aggregating in substrate network

work with current multipath routing and dimensioning algorithms because it is possible for these algorithms to split and route child demands through multipath. This chapter therefore proposes necessary adjustments in current network routing and dimensioning algorithms for constraining the actual aggregated demands to be routed through only one path as long as the substrate network still has enough resource and control the upper bound of all virtual link capacities to let the system guarantee QoS to customers.

The rest of this chapter is organized as follows, Section 3.1 describes the system. Section 3.2 discusses a single user on-off model. Section 3.3 calculates the actual resources for users' demands. Section 3.4 proposes a mathematical formulation of Mixed Integer Programming to route traffics through a substrate network. Numerical experiments are presented in Section 3.5. Finally, Section 3.6 concludes and discusses future work.

3.1 System description

With the assumption that virtual networks let the substrate network know the number of users connected to each link, the substrate network can multiplex many parts of virtual network demands with the same source destination address into an aggregated demand. And we can modularize the system as follows.

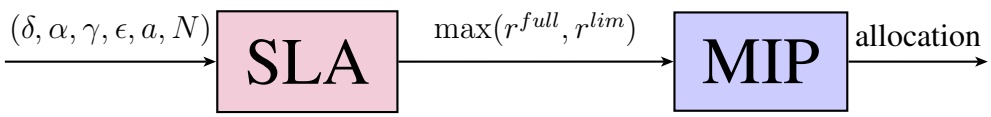

Figure 3.2: System functional description

Figure 3.2 shows the aggregated demand coming to a substrate network with the QoS parameters $1-\delta$ = probability of getting full availability, $1-\epsilon$ = probability of getting limited availability, γ = reduction factor which is the ratio of capacity provided to limited availability user comparing with full availability's. These parameters can be explained by the availability model as in Figure 3.3. where full availability is defined as the state in which the allocated

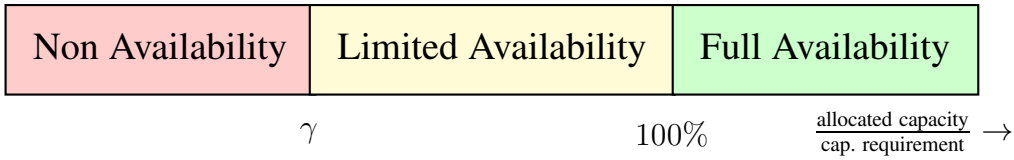

Figure 3.3: Availability model

bandwidth is at least equal to the requested bandwidth. Limited availability is defined as the state in which allocated bandwidth is smaller than or equal requested bandwidth but still greater than or equal γ times the requested bandwidth. Therefore, we have $\delta \geq \epsilon$ and if $\epsilon = 0$, then the customer will be guaranteed not getting non availability service. If $\epsilon \geq 0$, then sometime the customers have to suffer non availability service.

Beside QoS parameters, each aggregated demand also has the user parameters such as activity level α, peak rate a, and the number of users N in that aggregated demand. Service level agreement module calculates the necessary bandwidth for that demand as $\max\left(r^{full}, r^{lim}\right)$; where r^{full} is the minimum capacity for providing full availability and r^{lim} is the minimum capacity for providing limited availability. These values will be derived in Section 3.3. The system must guarantee that the allocated capacity is at least equal to this calculated bandwidth to uphold the SLA committed to customers. The SLA module then sends this calculated bandwidths as demands to the Mixed Integer Programming (MIP) module, which will push the traffic through the best route with the objective to minimize overall cost in hosting the traffic.

3.2 User model

In this chapter, following the assumption in [10] , the state of each user is modelled as a discrete-time Markov on-off process, as shown in Figure 3.4. The on-state represents the active state in which the user requires the constant peak rate a which does not vary with

each user. And the off-state represents the inactive state in which the user does not send any packets.

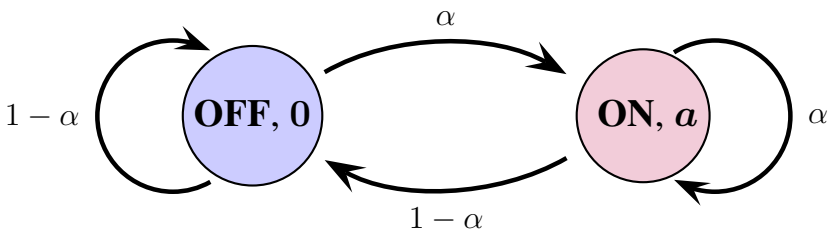

Figure 3.4: On-off user model

$$Pr\{0 < Q \leq a\} = 1 - Pr\{Q = 0\} = \alpha \tag{3.1}$$

where Q is bandwidth demand of the user. In this model, the on-off times of users have independent geometric distributions.

3.3 Service level agreement

This module automatically calculates the necessary capacity for aggregated demands from the their SLA parameters. Let Q^n denote the bandwidth demand of n^{th} user in the aggregated demand and let r^{full} denote the necessary capacity to host that aggregated demand. To guarantee the full availability, one must have the probability to get the full availability greater than $(1 - \delta)$, i.e

$$Pr\{\sum_n Q^n \leq r^{full}\} \geq 1 - \delta \tag{3.2}$$

The probability of n users to be active and $N - n$ users to be inactive in N independent users follows the binomial distribution. Hence, the term $\sum_n Q^n \leq r^{full}$ is equivalent to the statement that the number of concurrent active users cannot exceed $N_{excl}^{full} = \lfloor \frac{r^{full}}{a} \rfloor$, where r^{full} is the amount of capacities that the substrate network needs to provide to demands for

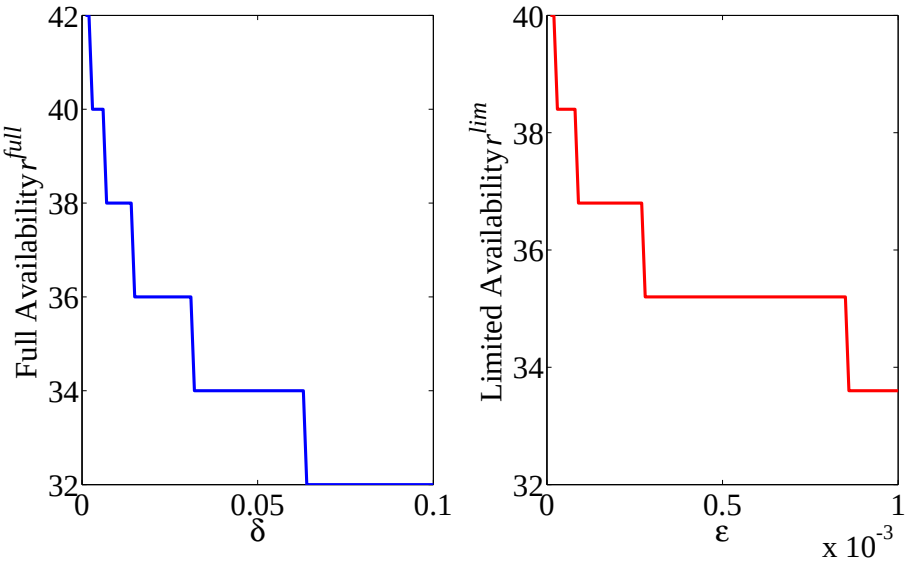

Figure 3.5: Minimum capacity requirement for full and limited availability; $N = 40$, $\alpha = .3$, $\gamma = .8$, $a = 2$.

guaranteeing the full availability service.

$$\sum_{n=0}^{N_{excl}^{full}} \frac{N!}{n!\,(N-n)!}\alpha^n(1-\alpha)^{N-n} \geq 1-\delta \tag{3.3}$$

The minimum capacity for providing full availability can finally be found:

$$r^{full} = aN_{excl}^{full} \tag{3.4}$$

With the same derivation, we can find out the minimum capacity requirement r^{lim} for the limited availability case with $N_{excl}^{lim} = \lfloor \frac{r^{lim}}{a} \rfloor$ as follows:

$$Pr\{\sum_{n} Q^n \leq r^{lim}\} \geq 1-\epsilon \tag{3.5}$$

$$\sum_{n=0}^{N_{excl}^{lim}} \frac{N!}{n!\,(N-n)!}\alpha^n(1-\alpha)^{N-n} \geq 1-\epsilon \tag{3.6}$$

$$r^{lim} = a\gamma N_{excl}^{lim} \tag{3.7}$$

As shown in Figure. 3.5, with the same number of users N, activity level α, reduction factor γ and user's peak rate a, we can vary δ and ϵ to have different r^{full} and r^{lim}. It is also shown that $r^{full} \geq r^{lim}$ does not always hold. For example, if we choose $\delta = 0.05$ and $\epsilon = 0.0002$, then we have $r^{full} = 34$ which is less than $r^{lim} = 37$. Because the allocated capacity has to

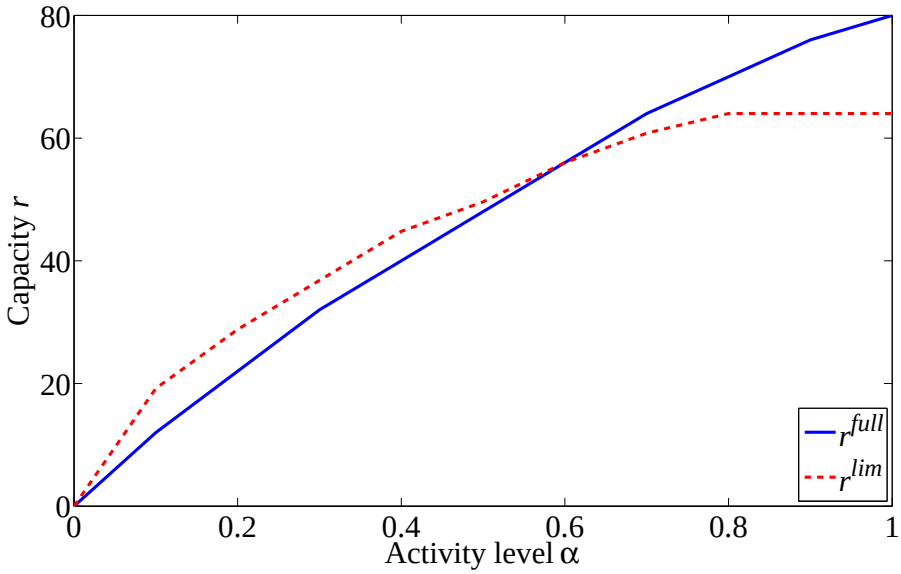

Figure 3.6: Minimum capacity requirement for full and limited availability vs source activity level α when $N = 40$, $\delta = 0.1$, $\epsilon = 0.0001$, $\gamma = 0.8$, $a = 2$.

satisfy both full and limited availability, it must be the maximum between r^{full} and r^{lim}:

$$r = \max\left\{r^{full}, r^{lim}\right\} \tag{3.8}$$

It is worth noting about the value of r^{full} and r^{lim} with respect to the activity level α. As shown in Figure 3.6, when α is small, $r^{full} \leq r^{lim}$. But when α is big, $r^{full} \geq r^{lim}$.

3.4 Mixed integer programming

The substrate network is modelled by the weighted indirect graph $G(V^s, E^s)$ where V^s is the set of all substrate nodes and E^s is the set of all substrate links each of which has index number l. We can calculate the cost p_l for one unit of traffic going through the substrate link with index l. This cost is varied with the cable distance and interface types. Here, the m-shortest path first algorithm [21] can be used to find out m least cost paths from any source to any destination in the substrate network. The link-path routing matrix with column representing the source-destination pair and row representing the substrate link is denoted by $\mathbf{H} = [H_{lj}]$ where $H_{lj} = 1$ if path j uses link l and $= 0$ otherwise; and $[H_{lj}^{i(k)}]$ is the routing matrix for virtual network k, demand i. The virtual network demand which has index k is modelled as a vector of demands. Each demand includes a source substrate node, a destination substrate node and the required capacity denoted by $r_i^{(k)}$ for i^{th} demand

of virtual network k. We denote $z_j^{i(k)}$ on path j as the amount of bandwidth that is assigned to demand i of virtual network k. Based on [42], the cost for one bandwidth unit of demand i of virtual network k on path j will be added by p_l if path j of demand i goes through link l, and added by 0 otherwise. Summing for all of link l, we have the cost as: $\sum_l H_{lj}^{i(k)} p_l$. Therefore, the total cost for $z_j^{i(k)}$ units of traffic going through path j is $z_j^{i(k)} \left(\sum_l H_{lj}^{i(k)} p_l \right)$ and summing for all k, i, j results in the total cost to route all virtual network demands: $\sum_k \sum_i \sum_j z_j^{i(k)} \left(\sum_l H_{lj}^{i(k)} p_l \right)$. Let $y_l^{(k)}$ be allocated bandwidth for link l of virtual network k. To minimize the total cost in the substrate network, we have to minimize the cost to route the traffic through the substrate network in parallel with the total bandwidth $\sum_l \sum_k y_l^{(k)}$ created for the virtual networks:

$$\sum_k \sum_i \sum_j z_j^{i(k)} \left(\sum_l H_{lj}^{i(k)} p_l \right) + \sum_l \sum_k y_l^{(k)} \tag{3.9}$$

The searching space is determined by constraints. We have the following capacity constraint: Firstly, the total flow going through any link of any virtual network must not exceed the link capacity of that virtual network. The virtual link capacity needed to host 1 flow unit will be increased by 1 unit at link l if this flow goes through link l. As a result, we have the virtual link bandwidth to host $z_j^{i(k)}$ flow expressible as $\sum_i \sum_j \sum_k H_{lj}^{i(k)} z_{lj}^{i(k)}$ and this virtual link bandwidth has to be smaller than the virtual link bandwidth $y_l^{(k)}$ that the substrate network provides to the virtual network. The second capacity constraint is the constraint on the substrate link capacity. Summing on link l of all virtual networks has to result in the added capacity not exceeding the bandwidth of the substrate link c_l:

$$\sum_k y_l^{(k)} \leq c_l \qquad \forall l \tag{3.10}$$

The third constraint is concerned with the demand condition. In particular, the summation of routed traffic has to be equal to the flow demand $r_i^{(k)}$:

$$\sum_j z_j^{i(k)} = r_i^{(k)} \qquad \forall i, k \tag{3.11}$$

The multipath routing is very good in diversifying a traffic through the network and helps the system to be robust with failures. However, since it cannot guarantee the quality of service for the users in aggregated demands, we only want to use it when single path routing cannot provide enough bandwidth to the demands. To build the constraint that prefers one path routing to multipath routing, we introduce a new indication variable $x_j^{i(k)}$ which has the

value of 1 if demand i of virtual network k goes through path j, and value of 0 otherwise. We construct the constraint as:

$$z_j^{i(k)} = x_j^{i(k)} r_i^{(k)} \qquad \forall i, j, k \tag{3.12}$$

The purpose of this constraint is to make one demand push all its traffic through only one path. Finally, we summarize our approach with the following optimization problem:

$$\text{minimize} \sum_k \sum_i \sum_j z_j^{i(k)} \left(\sum_l H_{lj}^{i(k)} p_l \right) + \sum_l \sum_k y_l^{(k)}$$

$$\begin{aligned}
\text{subject to} \quad & H^{(k)} z^{(k)} \leq y^{(k)} && \forall k \\
& \sum_k y_l^{(k)} \leq c_l && \forall l \\
& \sum_j z_j^{i(k)} = r_i^{(k)} && \forall i, k \\
& z_j^{i(k)} = x_j^{i(k)} r_i^{(k)} && \forall i, j, k \\
& z_j^{i(k)} \geq 0 && \forall i, j, k \\
& y_l^{(k)} \geq 0 && \forall l, k \\
& x_j^{i(k)} && \text{binary} \\
\text{variables} \quad & x_j^{i(k)}, y_l^{(k)}, z_j^{i(k)} &&
\end{aligned} \tag{3.13}$$

3.5 Results and discussions

All the experiments have been run on MATLAB with the setting as follows. The substrate network has the topology as in Figure 3.7 which is core network taken from [43]. There are two virtual network requests coming to the substrate network. The first one has eleven edge-to-edge connection demands and the second one has twelve edge-to-edge connection demands. Each connection demand has the source randomly chosen from the node set $\{6, 7, 8\}$ and the destination nodes from $\{11, 12, 13, 14\}$. Based on the number of users in each aggregated demand and the QoS parameters, we can find the actual bandwidth for the demands. The capacity of each substrate link is set to 1 Gbps for the experiments except the last one which is set randomly. The three shortest paths for each source-destination pair have been used to construct link-path routing matrix $\mathbf{H}$.

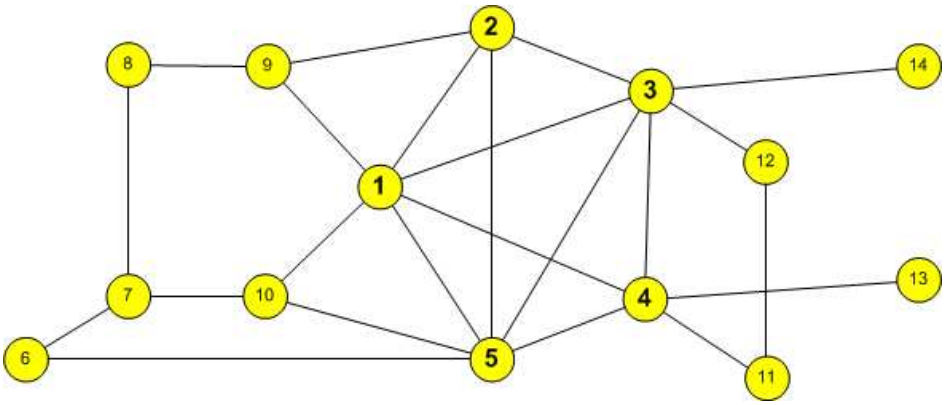

Figure 3.7: Experiment on the topology of Thailand's core network

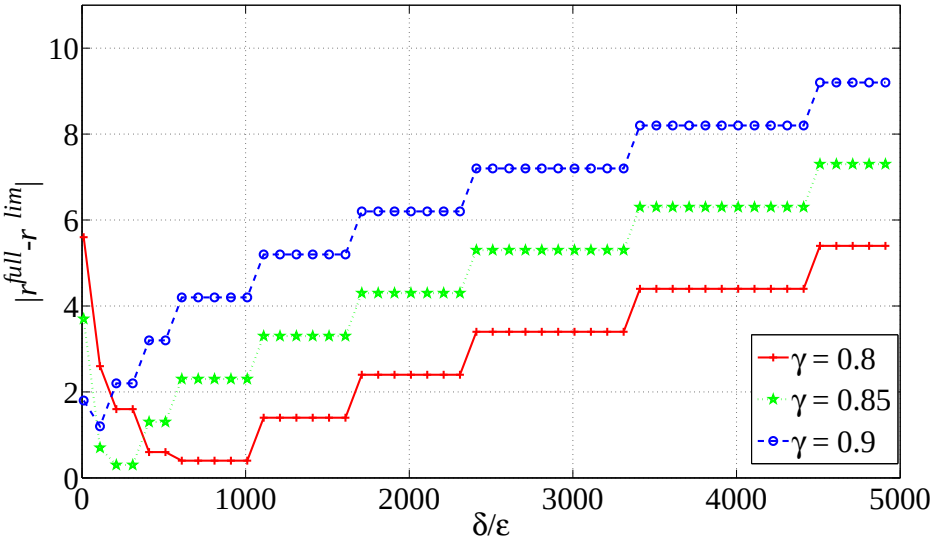

Figure 3.8: Offering SLA to minimize $|r^{full} - r^{lim}|$ ($\epsilon = 0.0001, \alpha = 0.3, a = 2, N = 20$)

Figure 3.8 depicts $|r^{full} - r^{lim}|$ vs $\frac{\delta}{\epsilon}$ when reduction factor γ varies from $\{0.8, 0.85, 0.9\}$. From the figure, the graph is stair with the increase of δ. If the increase of δ is not enough to make N_{excl}^{full} decrease, r^{full} as well as $|r^{full} - r^{lim}|$ remain unchanged and the graph is constant. If the increase of δ is big enough for N_{excl}^{full} to decrease one unit, then r^{full} decrease a units and $|r^{full} - r^{lim}|$ increases a and the graph has a jump. From this figure, the good enough interval that minimizes $|r^{full} - r^{lim}|$ is $\frac{\delta}{\epsilon} = [500, 1000]$. Based on the obtained results, we have chosen the QoS parameters for the following experiments: $\epsilon = 0.0001, \alpha = 0.3, a = 2, N = 20$.

Figure 3.9 presents the graph of mean cost per user vs activity level α when varying N in the case of unlimited substrate capacity. From the graph, the higher the number of users in an demand is the lower the cost per user becomes. The cost per user increases quite

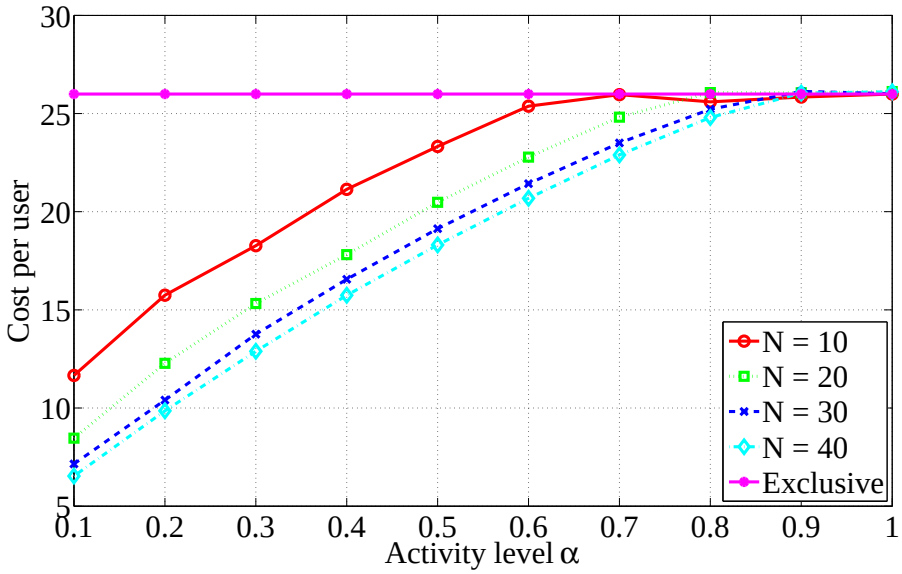

Figure 3.9: Cost per user vs activity level α with unlimited substrate link capacities; $\delta = 0.005$, $\epsilon = 0.0001$, $\gamma = 0.8$, $a = 2$.

linearly with activity level α. The cost when $\alpha = 0$ is zero and the cost when $\alpha = 1$ is equal to the cost in exclusive case. When N is small, the activity level to make cost maximize is small. When N is big we need big α for the cost reach maximum. Hence, the more users the demand has the more profit the InP gains. But we also see from the figure that the decrease of mean cost per user is not proportional with N. In particular, when N increases from 10 to 20 the mean cost per user decreases a lot. But, when N increases from 30 to 40, the mean cost per user decreases only a little bit.

Figure 3.10 shows the graph of mean cost per user vs activity level α when varying N in the case of limited substrate capacity. The substrate link capacity is limited by a number which is generated by a Gaussian random number generator with mean 250 and standard deviation is $1/10$ of the mean. From the figure, the bigger the demand is, the more expensive the mean cost per user becomes. The reason for this phenomenon is that the physical shortest path does not have enough bandwidth to host all traffic in a demand and this demand has to be split and routed through the worse path than the physical shortest path. The bigger the demand is the more the traffic of this demand has to be routed through the worse path. This causes the expensive mean cost per user in big demand.

Figure 3.11 demonstrates the total cost vs activity level α when varying the reduction factor γ in the case of unlimited substrate capacity. From the figure, the total cost is sensitive

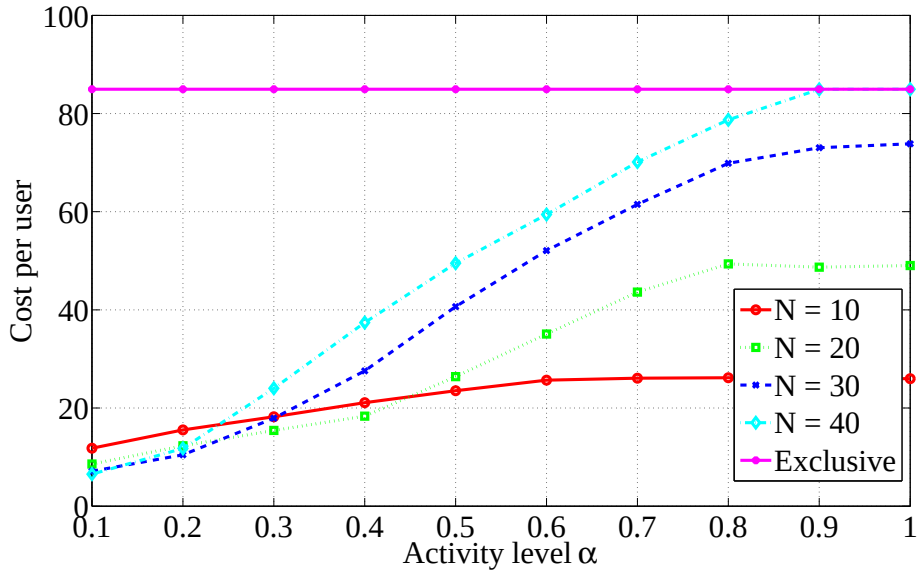

Figure 3.10: Cost per user vs activity level α with limited substrate capacity; $\delta = 0.005$, $\epsilon = 0.0001$, $\gamma = 0.8$, $a = 2$

with reduction factor γ when activity level α is small and not sensitive with γ when α is big. The reason for this phenomenon is as follows. From the Figure 3.6, when α is small the capacity to satisfy limited availability r^{lim} is bigger than the capacity to satisfy full availability r^{full}. When α is big, $r^{full} \geq r^{lim}$. The allocated capacity:

$$r = \max\left(r^{full}, r^{lim}\right) = \begin{cases} r^{lim} & \alpha \text{ is small} \\ r^{full} & \alpha \text{ is big} \end{cases} \tag{3.14}$$

The allocated capacity r is the main factor contributing to total cost. Note from (3.7) that γ only affects limited availability capacity. So, when α is small, $r = r^{lim}$ and total cost vary with γ. When α is big, $r = r^{full}$ and total cost does not vary with γ.

Figure 3.12 demonstrates the total cost vs activity level α when varying the reduction factor γ in the case of limited substrate capacity. From the figure, total cost varies with α and γ following the same trend as in unlimited substrate capacity case. The difference between Figure 3.12 and the Figure 3.11 is the total cost in limited substrate capacity case is higher than the total cost in unlimited substrate capacity case. The reason is that the substrate capacity of the shortest path is not enough to host the traffic flowing through that path. The overflow traffic has to be re-routed through the longer path. The cost of this overflow traffic increases the total cost.

Figure 3.13 depicts the histogram of the cost saving which is the ratio between our

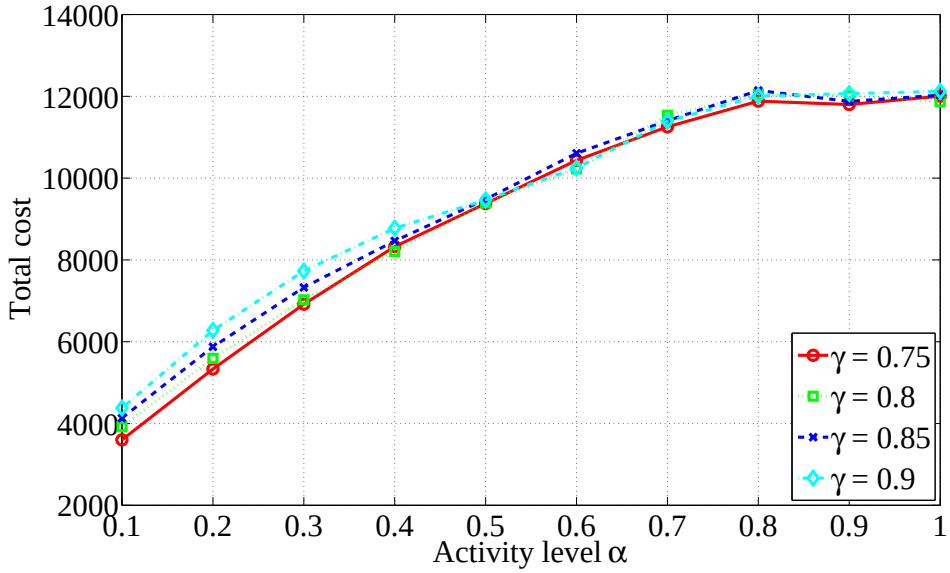

Figure 3.11: Total cost vs activity level α with unlimited substrate capacity; $\delta = 0.005$, $\epsilon = 0.0001$, $N = 20$, $a = 2$.

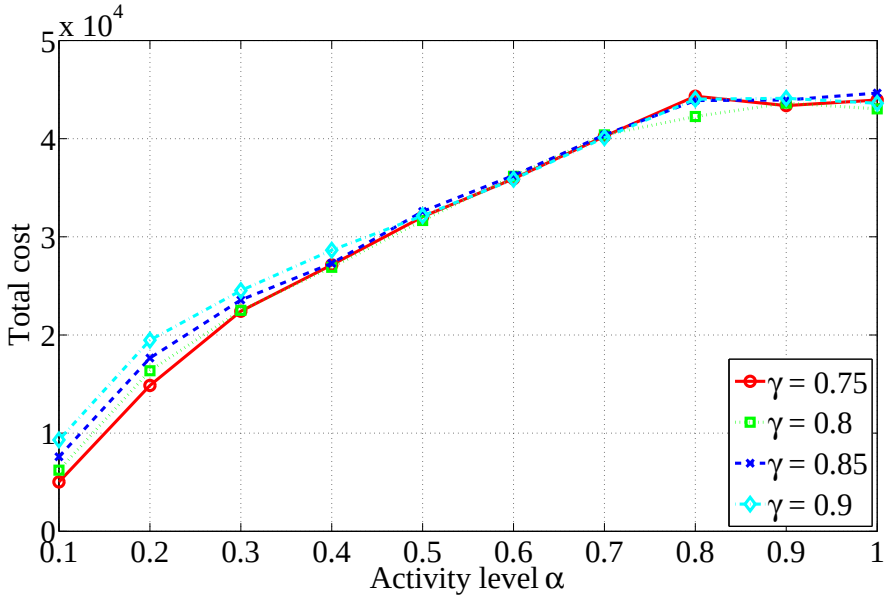

Figure 3.12: Total cost vs activity level α with limited substrate capacity; $\delta = 0.005$, $\epsilon = 0.0001$, $N = 20$, $a = 2$

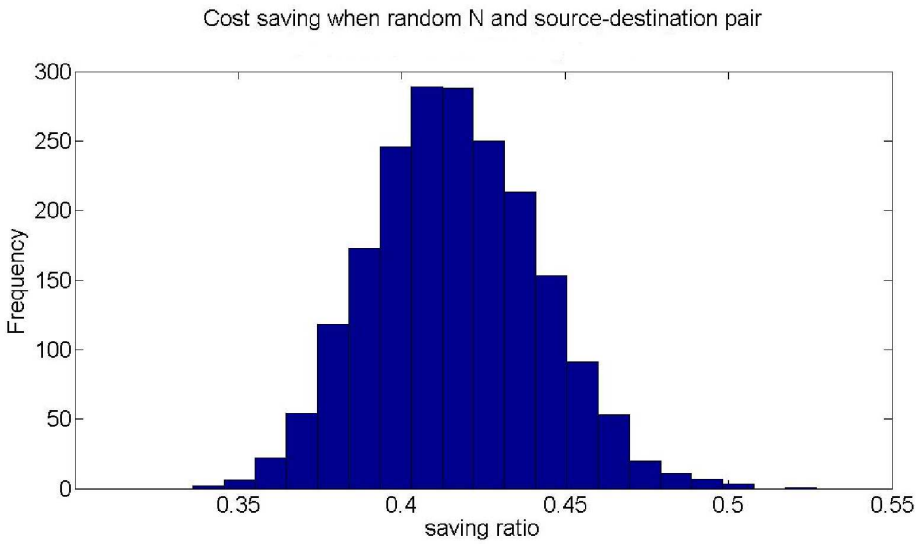

Figure 3.13: Cost saving histogram with many arrivals, each with random source, destination and number of users N. ($\delta = 0.05, \epsilon = 0.0001, \alpha = 0.3, \gamma = 0.8, a = 2$)

SLA cost and the exclusive cost in case of having many virtual network demand arrivals each with random source, destination nodes as well as random number of users N. From the figure, the cost saving ratio has mean of 0.4164 and standard deviation of 0.0257. This suggests that our method can save the resource of the substrate network significantly.

Figure 3.14 depicts total cost vs δ in the case of many arrivals each of which has the random source-destination pair nodes. The other parameters are set as $\epsilon = 0.0001, a = 2$. In this case, the substrate network resources are deduced with the amount required from virtual network demands. From the figure, the total cost decreases very fast when δ is from 0 to 0.01, but does not decrease any more when δ exceeds 0.01. The reason for this phenomenon is that, when δ reaches 0.01, the resource substrate network needed to provide to virtual network for guaranteeing the full availability service r^{full} is smaller than the resource substrate network needed to provide to the virtual network for guaranteeing the limited availability r^{lim}. In section 3.3, the necessary resource for a virtual network demand is $r = \max\left(r^{full}, r^{lim}\right)$. Hence, when $r^{full} \leq r^{lim}$, r does not change and equal to r^{lim} when δ continues to increase.

Figure 3.15 gives the ratio between the SLA cost and the exclusive cost when $\gamma = 1$, $\delta = 0$, $\epsilon = 0$. The ratio is calculated when randomizing all other experiment parameters such as δ randomly chosen from the interval [0.001, 0.01], ϵ from the interval [0.00005, 0.0115], α from the interval [0.1, 0.9], γ from the interval [0.7, 0.9], N from the interval

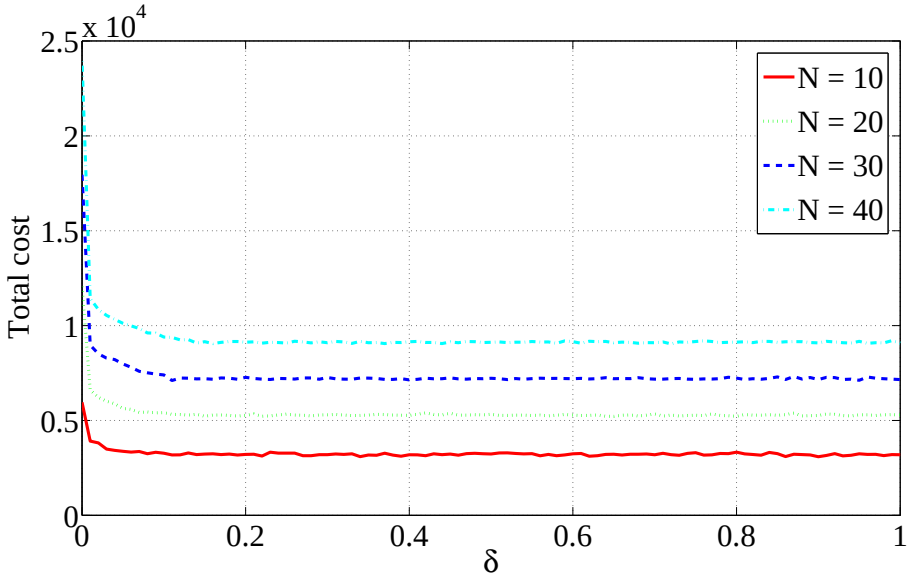

Figure 3.14: Total cost when varying δ in case of having many arrivals each has random source, destination

[15, 25], source nodes from the node set $\{6, 7, 8\}$, and destination nodes from the node set $\{11, 12, 13, 14\}$. The amount of cost saving has mean = 0.748, variance = 0.0479. This is the basic factor for the InPs to calculate how much discount they can offer to their customers who are willing to accept limited availability services.

3.6 Summary

This chapter formulates an optimal scheme for the InP to allocate resource to virtual networks. This scheme integrates the QoS features of resource allocation to routing and dimensioning optimization model for minimizing the embedding cost of virtual networks. The QoS features are defined by availability parameters. Each virtual-network customer who wants a service from the infrastructure provider will be provided only one SLA proposal including the percentage of time to get full availability, the percentage of time to get limited availability and the bandwidth reduction factor from full availability to limited availability. Based on the capacity that can be computed from the basic independent on-off source model assumption, we have formulated the optimization problem to push the virtual network demands through the best route for minimizing the cost of building all virtual networks.

This chapter's scheme is designed for running on the controller of multi-tenant data center network to provide additional levels of QoS to the traffic from server-hosting tenants.

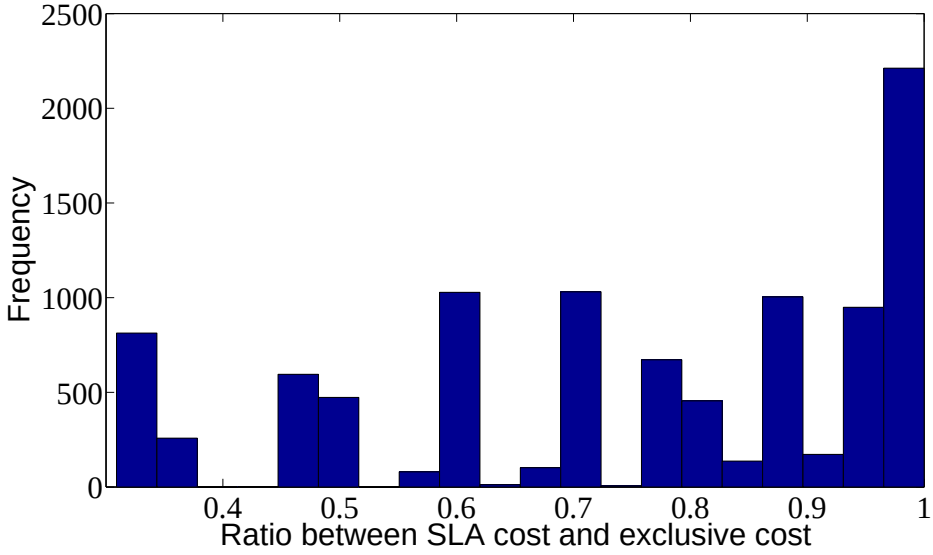

Figure 3.15: Cost saving when random $\delta, \epsilon, \alpha, \gamma, N$, and demands' source-destination

This traffic is best applicable for our algorithm whose properties are static, single path and centralized computation. A static resource allocation algorithm allocates resources to a demand only one time and does not change the allocated resources during the demand lifetime. The advantage of its stability fits well with low variation traffic from server-hosting tenants which are the aggregation traffic from many applications of tenants. For example, traffic from the server-hosting tenant, Google, is low variation because it is the aggregation traffic from many Google applications e.g gmail, youtube, google docs. A single path algorithm pushes all the traffic from an aggregated demand through only one best path. It has advantage of QoS with low jitter and high stability to support for strict QoS requirements of traffic from server-hosting tenants. A centralized algorithm is suitable to run on controller of multi-tenant data center network to solve the traffic optimization problem centrally with the advantage of fast convergence.

All the properties of this chapter's scheme are suitable to control the server-hosting traffic in multi-tenant data center network, but they are unsuitable to control the IaaS tenants' traffic which is dynamic, high variation and there are multi-path connections between the cloud servers. An example of IaaS service is Amazon virtual private cloud [34] which lets customers rent the computing and networking resources via its website. Customers can easily require additional resources when they need and release the resources back to the cloud when they finish using the resources. The flexibilities providing to customers make the Amazon virtual private cloud network very dynamic. Traffic sources in the Amazon virtual private

cloud are the servers, which send a huge amount of traffic when active and does not send when inactive. The behaviour of the traffic sources in Amazon virtual private cloud make its traffic property more pure chance than smooth. Furthermore, the topology of Amazon virtual private cloud is fat tree [44] to exploit the cheaper switches in the network core layer than in a conventional topology requiring high-density-port switches for interconnections within each data center. This fat tree topology has multi-path connections between cloud's servers.

With the basic setting of Amazon virtual private cloud in mind, in the next chapter, we decide to use optimization theory for deriving a new dynamic, multi-path algorithm to allocate resources optimally to virtual network demands. Our algorithm periodically re-adjusts the allocated bandwidth for all virtual networks' links. It can flexibly split the traffic through multi-path to improve the efficiency of substrate network resources. The next generation of Amazon virtual private cloud is proposed to be cloud resident data center, which uses OpenFlow system to provide additional network flexibilities to customers. This kind of cloud-evolved network virtualization has the large size in substrate network but has the small size in the virtual network. In the Chapter IV algorithm, we design the new model to have the centralized traffic control in each virtual network while having the distributed virtual link bandwidth control in the substrate network. Furthermore, the Chapter IV algorithm can use the SLA demand calculated from Chapter III in its constraints for guaranteeing the virtual network QoS even when the substrate network is congested.

CHAPTER IV

CENTRALIZED DYNAMIC VIRTUAL NETWORK ALLOCATION FOR CLOUD RESIDENT DATA CENTER

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 needed distributed control methods to scale with such a large network. The price for the scalability of the completely distributed control method at both the virtual layer and the substrate layer is the slow convergence of algorithm and the low stability of traffic. In this chapter, we argue that the distributed controls in both virtual and substrate networks are not necessary for the cloud resident data center environment because the cloud resident data center uses a 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 a very large network and is managed by only one organization. The distributed algorithm is stilled needed in the substrate network because the substrate network is a large with many global distributed data centers connected together and the substrate network may be owned and managed by many providers each of them has different administrative policy. Based on the specific properties of this model, we have used optimization theory to re-design the substrate algorithm for periodically re-adjusting the 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.

4.1 Introduction

The cloud resident data center [45], which has been proposed to replace e.g. the Amazon virtual private cloud (Amazon VPC) [34], 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 [44] as shown in Figure 4.1. 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 [45].

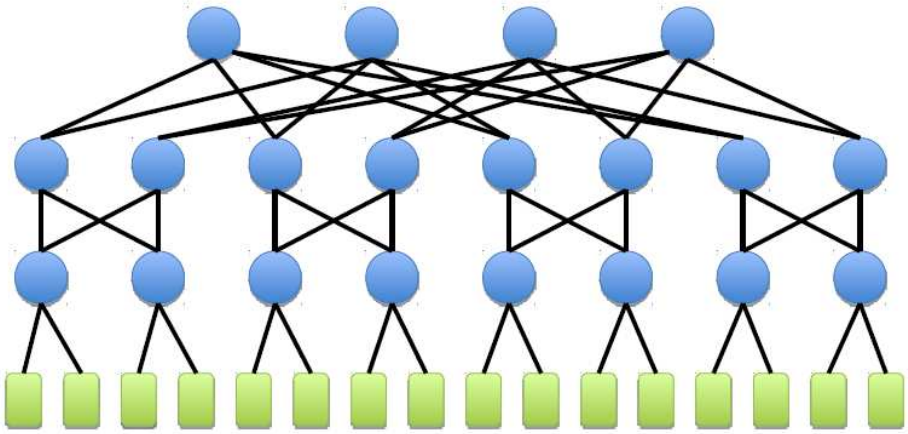

Figure 4.1: Fat tree topology

The multi-path load balancing problem has been tackled in [46] 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 [47] uses a marking system to decrease the source sending rate variation for achieving 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 [36]. 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 [28] 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 [31]. At the substrate layer, a periodically updated mechanism has been proposed to control the capacity of virtual links according to the derived congestion prices [29]. 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 chapter contains two original contributions.

Firstly, in this paper, the cloud resident data center testbed has been implemented in the OpenFlow [12] platform. We have decided to use the OpenFlow because of its centralized system structure whose controller can have a global 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 [12] as the physical layer, FlowVisor [1] as the network virtualization layer, and Network Operating System NOX [14] as the centralized controller. All the system modules communicate via the OpenFlow [12] protocol to emulate the dynamics of a small cloud resident data center testbed.

Secondly, this chapter re-designs the traffic control model in [28] 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 chapter is organized as follows. Section 4.2 describes the system model. Section 4.3 presents the derivation of our virtual link capacity updated algorithm for cloud resident data center. Section 4.4 demonstrates how we implement our algorithm in OpenFlow based testbed. Section 4.5 shows the results and discussions. The final section is Conclusion and our future works.

4.2 System description

Figure 4.2 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 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.

4.2.1 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.

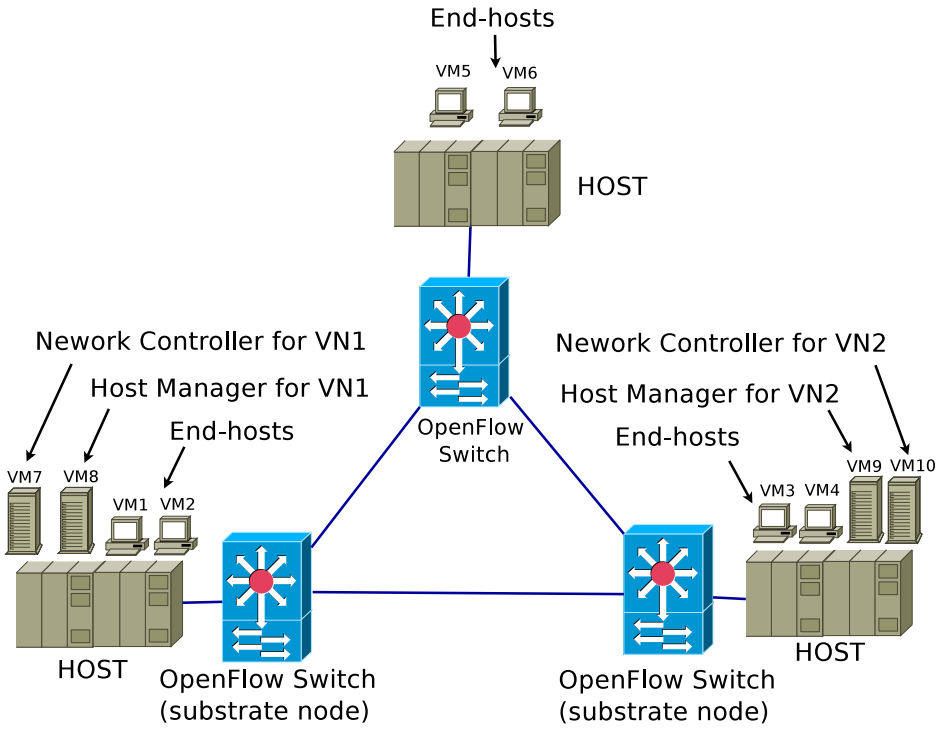

Figure 4.2: System functional description

4.2.2 Traffic control in the substrate network

The mission of traffic control in the substrate network is to adjust the virtual-link capacity periodically in order to maximize the total utility. In this chapter, 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 [28]. Instead, in this chapter, 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 [28] because the substrate nodes do not need the computational resource 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 [28]

4.3 Proposed optimization framework for cloud resident data center

4.3.1 Modeling and notations

The substrate network is modelled 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 Figure 4.3, there are three substrate nodes indexed by $\{1, 2, 3\}$ and six 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 modelled 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 $E^{v(k)}$ is the set of virtual links in virtual network k. For example, in Figure 4.3, there are two virtual networks, the yellow one and the red one, each with the same topology as that of substrate network.

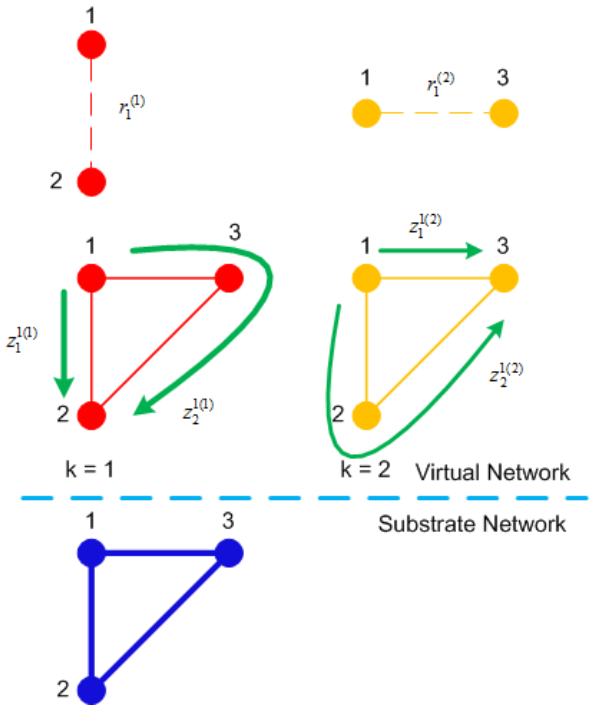

Figure 4.3: Example of traffic allocation when hosting many virtual networks on a substrate network.

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 Figure 4.3, there is one demand on each of the

two virtual networks, denoted as $r_1^{(1)}$ and $r_1^{(2)}$, respectively. In Figure 4.3, 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 instantaneous 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 [21] 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 [23] 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 Figure 4.3, 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 $\mathbf{H}^{(2)} = [1\ 0; 0\ 0; 0\ 1; 0\ 0; 0\ 1; 0\ 0]$.

4.3.2 Virtual-network centralized algorithm

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

$$\begin{aligned} \text{maximize:} \quad & U^{(k)}\left(z_j^{i(k)}, y_l^{(k)}\right) \\ \text{subject to:} \quad & \sum_i \sum_j H_{lj}^{i(k)} z_j^{i(k)} \leq y_l^{(k)} \quad \forall l \in L_k \\ & z_j^{i(k)} \geq 0 \qquad\qquad\qquad\qquad \forall i, j \\ \text{variables:} \quad & z_j^{i(k)} \end{aligned} \tag{4.1}$$

As in [28], the objective function $U^{(k)}\left(z_j^{i(k)}, y_l^{(k)}\right)$ of the virtual network centralized problem (4.1) can be any concave utility function which varies with the capacity allocation parameters. The first constraint implies for each virtual network that the instantaneous 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. The two examples of virtual network centralized problem are throughput sensitive problem and delay-sensitive problem.

4.3.3 Substrate-network distributed algorithm

Our starting point is the same as [28] 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:

$$\begin{aligned} \text{maximize:} \quad & \sum_k w^{(k)} U^{(k)}\left(z_j^{i(k)}, y_l^{(k)}\right) \\ \text{subject to:} \quad & \sum_i \sum_j H_{lj}^{i(k)} z_j^{i(k)} \leq y_l^{(k)} \quad \forall l \in L_k, \forall k \\ & \sum_k y_l^{(k)} \leq c_l \qquad\qquad\qquad\qquad \forall l \in L \\ & y_l^{(k)}, z_j^{i(k)} \geq 0 \qquad\qquad\qquad\qquad \forall i, j, k \\ \text{variables:} \quad & z_j^{i(k)}, y_l^{(k)} \end{aligned} \tag{4.2}$$

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 [28].

The problem in (4.2) is the convex problem because the objective function of this problem is the positive-weighted sum of concave objective functions (4.1) 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 [31]. 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 (4.1). The optimization solver CVXOPT [37] which implements the so called primal-dual interior point method [29] is used to solve each problem of (4.1) 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:

$$\begin{aligned} \text{maximize:} \quad & \sum_k w^{(k)} U^{(k)} \left(z_j^{*i(k)}, y_l^{(k)} \right) & \\ & \sum_k y_l^{(k)} \leq c_l & \forall l \in L \\ & y_l^{(k)} \geq 0 & \forall k, l \\ \text{variables:} \quad & y_l^{(k)} & \end{aligned} \tag{4.3}$$

The above master optimization problem is also the convex problem because it has the same concave objective function as (4.2) and the affine constraints. As suggested in [29] 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 (4.3) 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 [31] to decouple it. We find the Lagrange function of (4.3) as follows:

$$\begin{aligned} \mathcal{L}\left(y_l^{(k)}, \mu_l \right) &= \sum_k w^{(k)} U^{(k)} \left(z_j^{*i(k)}, y_l^{(k)} \right) - \sum_l \mu_l (\sum_k y_l^{(k)} - c_l) \\ &= \sum_k w^{(k)} U^{(k)} \left(z_j^{*i(k)}, y_l^{(k)} \right) - \sum_k \sum_l y_l^{(k)} \mu_l + \sum_l \mu_l c_l \end{aligned} \tag{4.4}$$

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

We use the gradient projection method to find the supremum of problem (4.4) which has implicit constraint of non-negative orthant and the variable of $y_l^{(k)}$. The update function for $y_l^{(k)}$ is

$$y_l^{(k)}(t+T) = \left[y_l^{(k)}(t) + \beta \frac{\partial}{\partial y_l^{(k)}} \mathcal{L}\left(y_l^{(k)}, \mu_l\right) \right]^+ \tag{4.5}$$

where T 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.

The partial derivative of objective function at the optimality is the link congestion price. Apply to our problem:

$$\frac{\partial}{\partial y_l^{(k)}} \left(\sum_{k \in K} U^{(k)}\left(z_j^{*i(k)}, y_l^{(k)}\right) \right) = \lambda_l^{*(k)} \tag{4.6}$$

Using (4.6) we have partial derivative of Lagrange function as

$$\frac{\partial}{\partial y_l^{(k)}} \mathcal{L}\left(y_l^{(k)}, \mu_l\right) = w^{(k)} \lambda_l^{*(k)} - \mu_l \tag{4.7}$$

To find μ_l, we define $p^{*(k)}$ as the optimal value of (4.1) and p^* as the optimal value of (4.3). We have

$$\begin{aligned} p^* &= \sum_k w^{(k)} p^{*(k)} \\ \frac{\partial p^*}{\partial c_l} &= \sum_{k \in K_l} w^{(k)} \frac{\partial p^{*(k)}}{\partial c_l} \\ \mu_l^* &= \sum_{k \in K_l} w^{(k)} \lambda_l^{*(k)} \frac{\partial y_l^{(k)}}{\partial c_l} \end{aligned} \tag{4.8}$$

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 the constraint $c_l = \sum_{k \in K_l} y_l^{(k)}$, we allocate fairly all of our residual capacity to virtual network customers:

$$\frac{\partial y_l^{(k)}}{\partial c_l} = \frac{y_l^{(k)}}{\sum_{\kappa \in K_l} y_l^{(\kappa)}} \tag{4.9}$$

Hence, we have from (4.8), (4.9) that

$$\mu_l^* = \frac{\sum_{k \in K_l} w^{(k)} \lambda_l^{*(k)} y_l^{(k)}}{\sum_{k \in K_l} y_l^{(k)}} \tag{4.10}$$

And the final virtual link capacity update function can be obtained by combining (4.5) (4.7)(4.10) as follows

$$y_l^{(k)}(t+T) = \left[y_l^{(k)}(t) + \beta \left(w^{(k)} \lambda_l^{*(k)} - \frac{\sum_{k \in K_l} w^{(k)} \lambda_l^{*(k)} y_l^{(k)}(t)}{\sum_{k \in K_l} y_l^{(k)}(t)} \right) \right]^+ \tag{4.11}$$

As a summary, in (4.11), we can find the value of $y_l^{(k)}$ at time $t+T$ from the previously updated value of $y_l^{(k)}$ at time t. In the recursion, β is a constant step size 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.

4.3.4 Convergence and optimality

Theorem: The updated equation (4.11) converges to optimal point of (4.2) if:

- The problem (4.1) is convex optimization.
- The virtual link bandwidth allocation is updated after the convergence of (4.1)
- The stepsize β in (4.11) is small enough or diminishing (diminishing step size usually converge faster constant step size).

Proof:

From the above interpretation of (4.11) 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. 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 $\hat{y}_l^{(k)}$. At the limited point we have:

$$w^{(k)}\lambda_l^{*(k)} = \frac{\sum_{k\in K_l} w^{(k)}\lambda_l^{*(k)}\hat{y}_l^{(k)}}{\sum_{k\in K_l}\hat{y}_l^{(k)}} \tag{4.12}$$

It means that at the limited point $\hat{y}_l^{(k)}$, 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 any more. Hence,

$$\begin{aligned}
\frac{\partial}{\partial y_l^{(k)}}\mathcal{L}(\hat{y}_l^{(k)}, \mu_l) &= 0 \qquad \forall l, k \\
\sum_k y_l^{(k)} &\leq c_l \qquad \forall l \\
\mu_l(\sum_k y_l^{(k)} - c_l) &= 0 \qquad \forall l
\end{aligned}$$

The KKT conditions [29] are satisfied for the convex problem of (4.3), then the limited point of (4.11) is the optimal point of (4.3). Because the equation (4.1) is converged before the virtual link bandwidth update time, at $y_l^{(k)} = \hat{y}_l^{(k)}$, 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 (4.2).QED

4.4 Implementation

4.4.1 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.

4.4.2 Logical model

The logical model of our network is presented in Figure 4.4. In this model, three substrate nodes have been created by installing OpenFlow switch software v$0.8.9$r2 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 [38] 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 CVXOPT [37] to calculate the optimal rates for all end-hosts and congestion price values for all virtual links in the virtual network.

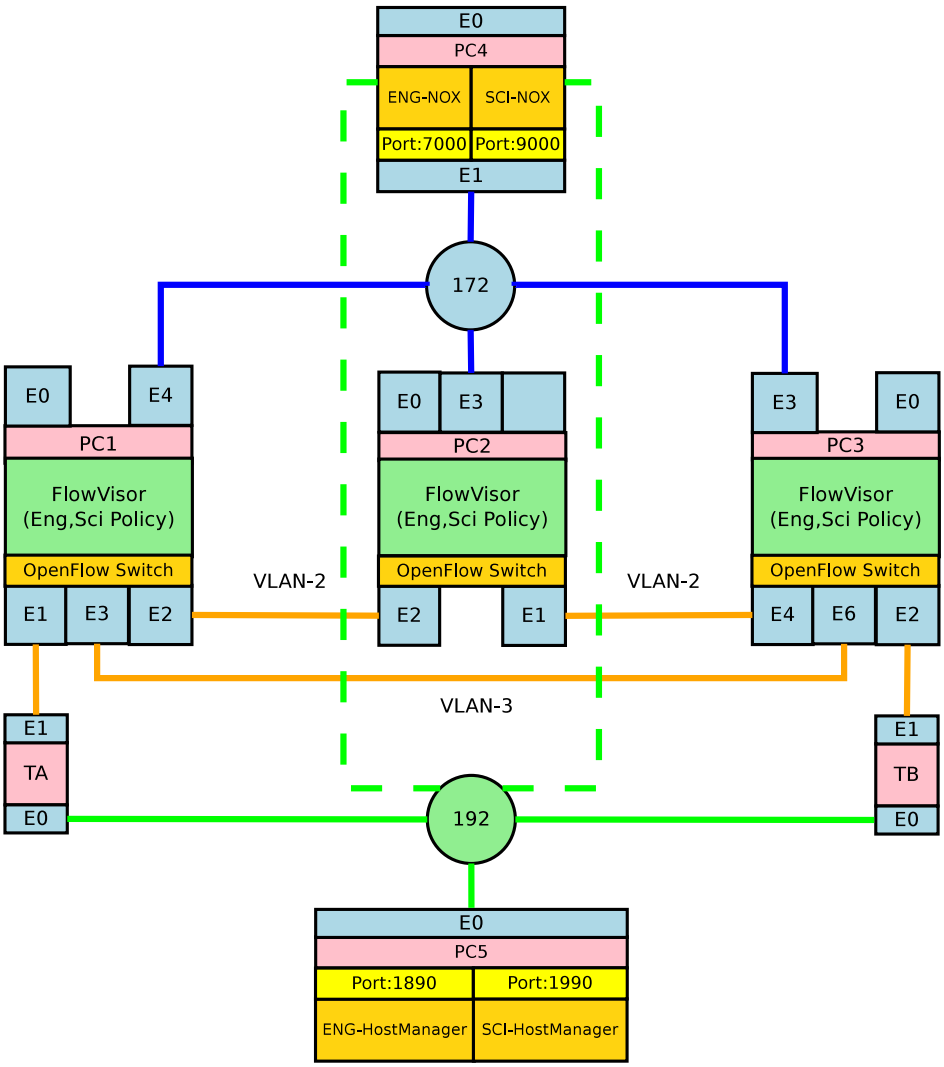

Figure 4.4: Logical model of cloud resident data center testbed in this chapter.

There are three networks used to connect all elements of system as depicted in Fig-

ure 4.4. 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.

4.4.3 Virtual-network centralized algorithm implementation

We have implemented the throughput-sensitive centralized algorithm on each host manager using Python CVXOPT [37] with the objective function which tries to maximize the overall throughput of network meanwhile keeps the link capacity uncongested as follows:

$$\texttt{max:} \sum_i w_i^{(k)} \log\left(\sum_j z_j^{i(k)}\right) - q \sum_{l \in L_k} \exp\left(u_l^{(k)}\right) \tag{4.13}$$

where the utilization $u_l^{(k)}$ of virtual network 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 here is different from [28] in that we add weight $w_i^{(k)}$ to each demand for controlling the network resource provided to each demand. This weight is calculated by $w_i^{(k)} = \frac{r^{(k)}}{\sum_i r^{(k)}}$, where $r^{(k)}$ 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 [38] 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 like this $(100, 40)$, then it understands that it should send its 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 $100 * T$ packets/s with VLAN-ID = 2 as well as $40 * T$ packets/s with VLAN-ID = 3 and send them out to switch PC1 as fast as possible. After sending out all the created packets, end-host will stop for the rest of period T. 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.

4.4.4 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 (4.11). 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.

4.4.5 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 [28] 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:

$$\texttt{min:} \sum_i \sum_j z_j^{i(1)} \sum_l H_{lj}^{i(1)} (p_l + f(u_l^{(1)})) \tag{4.14}$$

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.

4.5 Evaluation

4.5.1 Simulation results

The Figure 4.5 compares the time to convergence between our proposed algorithm (CD: Centralized control in virtual networks, Distributed control in the substrate network) and Davinci algorithm [28] (DD: Distributed control in both virtual networks and substrate network) in two nodes topology. The upper graph of Figure 4.5 shows that with the same setting of $q = 0.5, w^{(1)} = 1, w^{(2)} = 10^5$ $\beta = 0.2$ and $r_1^{(1)} = 110$ Mbps, our proposed

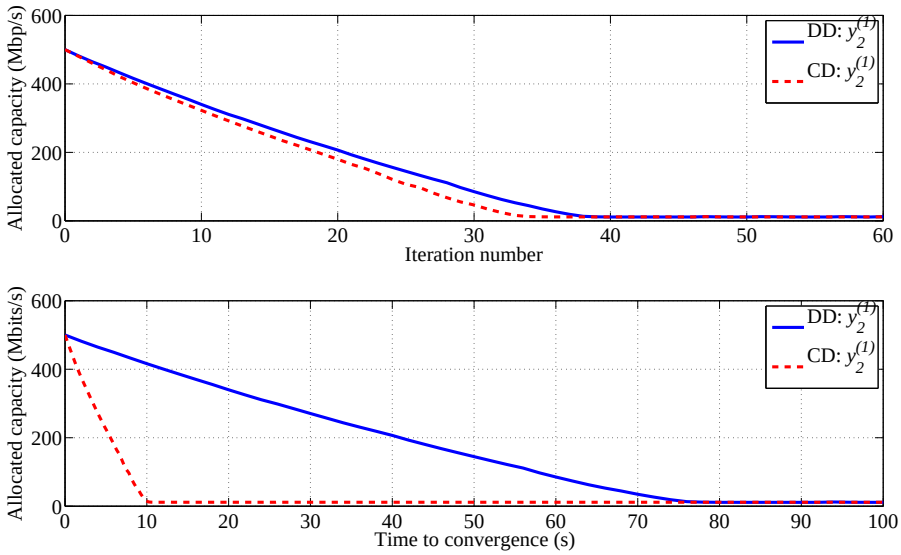

Figure 4.5: Comparing with Davinci in two nodes topology. The Simulation running with MATLAB on computer of 2 duo CPU T9400@2.53GHz, 4GBytes of RAM, Ubuntu 12.04

algorithm needs nearly the same number of outer iterations, which are the number of iterations to solve the master problem (4.3) distributedly, as in [28]. But, as showed in the lower graph, our algorithm needs much shorter time to converge than the one in [28]. 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 [28] needs 2 seconds to solve them distributedly. The more time needed for [28] to solve problem at each iteration make it converge slower than our proposed algorithm.

The Figure 4.6 presents the number of iterations to convergence of our algorithm in the Abilene topology [48]. From the figure, the algorithm converges very slow when step size β is so small and diverges when step size β is large. The reason is that when β 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.

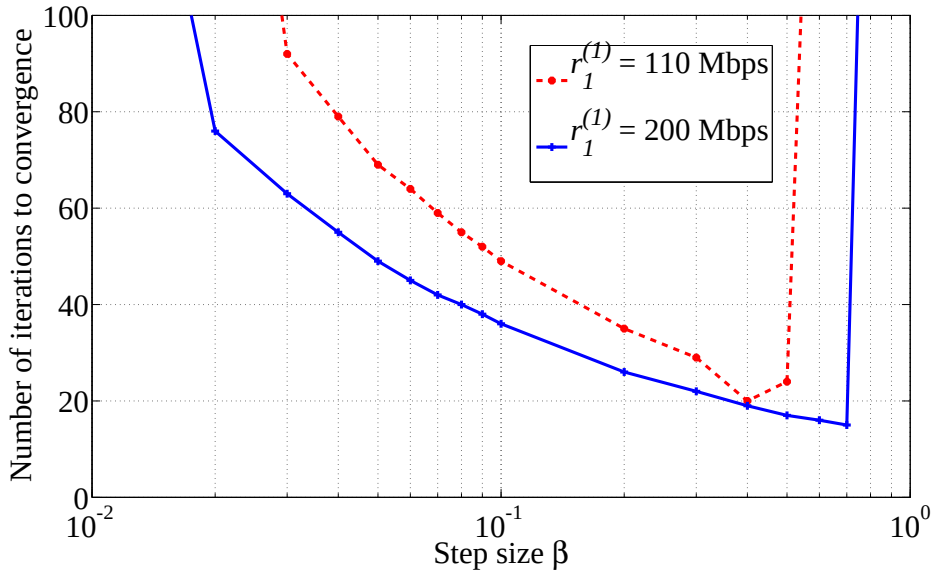

Figure 4.6: Step size β vs number of iterations to convergence in Abilene topology

4.5.2 Testbed evaluation results

Figure 4.7 depicts the dynamics of actual traffic on ENG virtual network (upper graph) and SCI virtual network (lower graph). This result is based on the demands which are changed randomly every 40 optimization iterations and β 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 β. 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, then our algorithm needs more iterations to converge.

Figure 4.8 depicts the virtual link updated period T 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 host manager of that virtual network which is equal 0.7 in our case, then our algorithm will need very long time to converge. If $T \geq 0.7$ second, then the virtual network optimization problem is guaranteed to be solved completely before the updated time of our proposed algorithm (4.11), and our algorithm is converged. When our proposed algorithm converge,

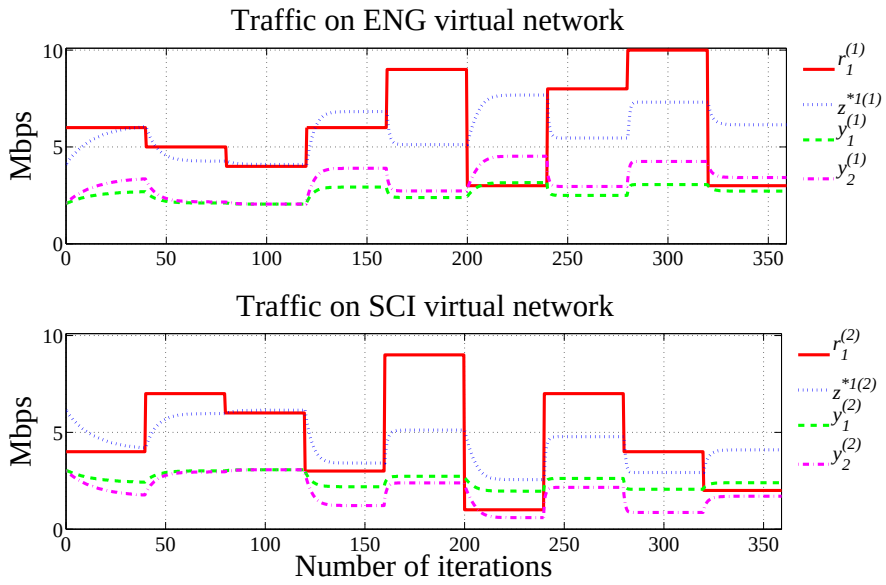

Figure 4.7: Dynamics of traffic on ENG and SCI network

the time to convergence is quite linear with the iteration period (T). 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.

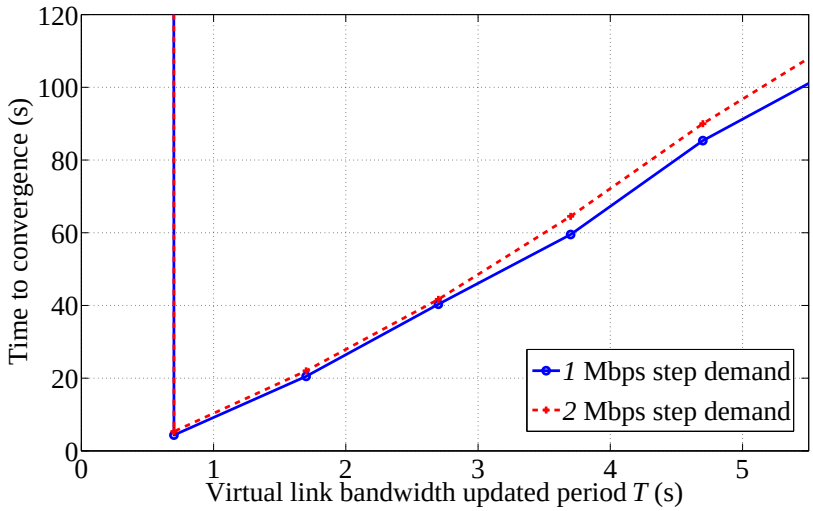

Figure 4.8: Time to convergence vs virtual link bandwidth updated period T

Figure 4.9 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 equation (4.11) 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 allocated the equal bandwidth 3.5 Mbps to each virtual link hosted on the physical link VLAN 3. This initial allocation phenomenon caused by fair allocation of equation (4.9). After the initial allocation, our algorithm gradually adjusts the allocated capacity to the new optimal point which allocated total 3 Mbps of traffic for the virtual network ENG and 7 Mbps of traffic for the virtual network SCI.

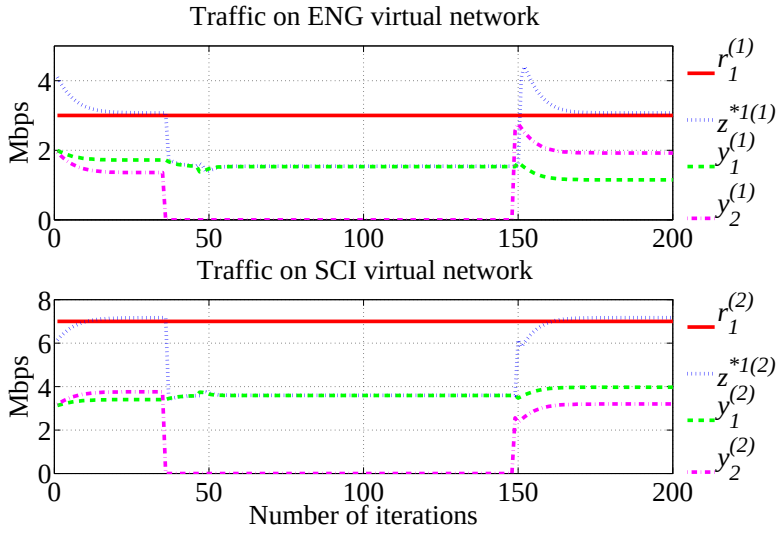

Figure 4.9: Traffic dynamics when the physical link VLAN 3 is down and then recovered

4.6 Conclusion

In this chapter, 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 converges to the optimality faster than the previous approach such as Davinci [28] so it can use physical resource 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 pre-calculating 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 comming to the network, the host manager only needs to find the optimal results from the mapping table. Future results will be reported accordingly.

CHAPTER V

CONCLUSION

This dissertation aims to develop the novel network control algorithms for working appropriately in the recent network virtualization of multi-tenant data center network. Taking into account the new network virtualization constraints, we have originally derived the two algorithms, static and dynamic, by using optimization theory. The in-depth investigations of the two algorithms are presented in **Chapter** III and **Chapter** IV, correspondingly. Summary of the contributions of each chapter is shown in the following sections, together with suggestions for possible future works.

5.1 Contributions from chapter III

In **Chapter** III, an optimization problem of resource allocation for network virtualization has been formulated and solved. The novelty in our formulation is that we have integrated the QoS features of the resource allocation to routing and dimensioning optimization model for minimizing the operating cost of virtual networks. The QoS is defined by availability parameters. Each customer, who wants a service from the InPs, is provided a SLA proposal. This proposal includes the percentage of time to get full availability, the percentage of time to get limited availability and a bandwidth reduction factor from full availability to limited availability. Based on the capacity, which can be computed from the basic independent on-off source model assumption, this thesis formulates the optimization problem to push the virtual network demand through their best route for minimizing the cost of building all virtual networks. The cost saving ratio being quantitative here with mean = 0.748 and variance = 0.0479 based on reported numerical results of Figure 3.15 is significant. The InP can offer a portion of that obtainable saving to limited availability customers, and they will get the benefit from the reduced service price. Furthermore, the InP which has additionally flexible services can attract more customers than the InP with only exclusive service.

5.2 Contributions from chapter IV

In **Chapter** IV, the cloud resident data center testbed has been implemented in the OpenFlow platform. This chapter has decided to use the OpenFlow because of its centralized system structure whose controller can have a global 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. In this chapter, A testbed has been implemented on Linux-based computers installed with OpenFlow switch [12], as the physical layer, FlowVisor [1] as the network virtualization layer, and Network Operating System NOX [14] as the centralized controller. All the system modules communicate via the OpenFlow [12] protocol to emulate the dynamics of a small cloud resident data center testbed.

Secondly, this chapter uses optimization theory [29] to re-design the traffic control model in [28] for the OpenFlow-based network setting of cloud resident data center. Due to the centralized control structure of the OpenFlow, we have applied [31] to re-derive the optimization problem into the centralized algorithm instead of the distributed algorithm for each virtual network. In the substrate network, we have derived the new distributed algorithm to re-allocate bandwidth for the virtual links periodically. In addition, we have theoretically proven the convergence property of this algorithm. In the experiment, we have implemented our new algorithm on both simulation (MATLAB) and testbed (OpenFlow network virtualization system). The simulation results show that our algorithm converges faster than the previous algorithms such as Davinci [28]. The testbed's experiment results show that our algorithm converges fast to re-adjusts effectively the traffic flowing in cloud resident data center in both cases, normal and failure recovery case.

5.3 Possible future works

- **Malicious behaviour**. Same as [28], the algorithm derived in **Chapter** IV also cannot provide substrate resources fairly if a virtual network tries to acquire more resources than necessary. In this situation, the greedy behaviours of any virtual network will harm the performance of all other virtual networks. The paper [32] has proposed a non-cooperative game framework in which a new term in congestion price has been introduced for charging additional money to all the virtual networks hosting on the sub-

strate congested links. This new formulation can be applied to this research's framework to relieve the malicious behaviours.

- **Relax capacity assumption.** When solving the optimization problem in **Chapter** IV, we have assumed that InP allocates fairly all of its substrate resources to virtual network customers. This assumption limits the searching space on the boundary of the constraint set to prove theoretically that the algorithm derived in this thesis always converges to the optimal point. But now a day, because many providers set maximum rate for each traffic demand at their access switches, the total bandwidth of all demand may not reach the physical capacity and make the assumption invalid. In this situation, the algorithm needs additional constraints to limit the resources allocated for each demand and the searching space is not only on the boundary but also inside the constraint set. Searching inside the constraint set may make the algorithm not converge. An interesting future direction of this dissertation is to improve the algorithm for the convergence even in this practical situation.

- **Reduction of substrate computing load.** To calculate the virtual link capacity for all virtual networks, the substrate algorithm derived in **Chapter**IV spends a lot of computing resources. Following [49], redesign the substrate algorithm can be redesigned to calculate only virtual network's congestion prices, then update back to virtual network's controllers. The controllers will base on received congestion price to figure out the needed virtual link bandwidth. Reducing from calculating all virtual link capacities to only congestion prices unloads the substrate's computing tasks significantly

- **Scaled-up experiment.** In this dissertation's algorithm, many important parameters need to be found from experiments such as step size β and updated period T. The recommended values of these parameters are only trustable if the algorithm can be tested in the big enough testbed. With the limitation of our research lab resources, the algorithm has just been implemented in a small computer based testbed. Fortunately, many inter-continental future internet testbeds such as OF@TEIN, OF@JGN, PlanetLab, VINI, FEDERICA [11] have been rapidly built up. They use the same software modules e.g. OpenFlow, OpenVswitch, FlowVisor as in this research's testbed to implement their inter-continental testbed. Implementing the algorithm in the inter-continental testbed and find the best way to tune the algorithm's parameters is one of

interesting future direction of this dissertation.

- **Node assigment.** Virtual network allocation needs both links and nodes assignment. Specially for cloud network, the efficient allocation of computing nodes is more important than the efficient allocation of link bandwidth. The dynamics in node assignment relate to adding more computing nodes for virtual data center or taking them back to the substrate cloud. Hence, the node assignment algorithm does not need as fine scale control as virtual link bandwidth assignment. One interesting extension for our algorithm is to allocate optimally both nodes and links for virtual network demand.

- **TCP compatibility.** Currently, the algorithm in this thesis can only work with the optimization derived traffic control algorithm in virtual network. Because the popular TCP protocol, which is designed by heuristics in the past, can be derived from optimization decomposition [49], it's likely that our algorithm is able to work with TCP. An interesting future work to develop our algorithm is to make it work with TCP. Compatibility with TCP will enable our algorithm to integrate to many existing systems such as cloud network and internet.

- **Improve convergence rate.** Speed up the convergence rate of this thesis's algorithm is one of the ultimate goal. We have changed the control model in each virtual network from distributed to centralize; we have tuned step size β and updated period T to make our algorithm converge faster. Beside that, another way to improve the convergence rate of virtual network algorithm is pre-calculating 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.

References

[1] Sherwood, R., Gibb, G., Yap, K., Appenzeller, G., Casado, M., McKeown, N., and Parulkar, G., Flowvisor: A network virtualization layer. Tech. Rep. Openflow-tr-2009-1, Stanford University, 2009.

[2] Haider, A., Potter, R., and Nakao, A., Challenges in resource allocation in network virtualization. Proceedings of 20th ITC Specialist Seminar on Network Virtualization 18 (2009): 20.

[3] Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I., and Warfield, A., Xen and the art of virtualization. ACM SIGOPS Operating Systems Review 37 , 5 (2003): 164–177.

[4] Comer, D. Internetworking with TCP/IP Vol. I: Principles, Protocols, and Architecture. Prentice hall Englewood Cliffs, NJ, 1995.

[5] Kivity, A., Kamay, Y., Laor, D., Lublin, U., and Liguori, A., KVM: The Linux virtual machine monitor. Proceedings of the Linux Symposium 1 (2007): 225–230.

[6] Awduche, D. and Agogbua, J., Requirements for traffic engineering over MPLS., Available from: `http://search.ietf.org/rfc/rfc2702.txt`, 1999.

[7] IEEE Standard for Local and metropolitan area networks–Media Access Control (MAC) Bridges and Virtual Bridged Local Area Networks–Corrigendum 2: Technical and editorial corrections. IEEE Std 802.1Q-2011/Cor 2-2012 (Corrigendum to IEEE Std 802.1Q-2011) (2 2012): 1 -96.

[8] Brackett, C. Dense wavelength division multiplexing networks: Principles and applications. IEEE Journal on Selected Areas in Communications 8 , 6 (1990): 948–964.

[9] Lua, E., Crowcroft, J., Pias, M., Sharma, R., and Lim, S. A survey and comparison of peer-to-peer overlay network schemes. IEEE Communications Surveys and Tutorials 7 , 2 (2005): 72–93.

[10] Fiedler, M. On resource sharing and careful overbooking for network virtualisation. International Journal of Communication Networks and Distributed Systems 6 , 3 (2011): 232–248.

[11] Chowdhury, N. and Boutaba, R. A survey of network virtualization. Computer Networks 54 , 5 (2010): 862–876.

[12] McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., Shenker, S., and Turner, J. OpenFlow: enabling innovation in campus networks. ACM SIGCOMM Computer Communication Review 38 , 2 (2008): 69–74.

[13] The OpenFlow Switch Specification., `http://OpenFlowSwitch.org/`.

[14] Gude, N., Koponen, T., Pettit, J., Pfaff, B., Casado, M., McKeown, N., and Shenker, S. NOX: towards an operating system for networks. ACM SIGCOMM Computer Communication Review 38 , 3 (2008): 105–110.

[15] Drwal, M. and Gasior, D., Utility-based rate control and capacity allocation in virtual networks. Proceedings of the 1st European Teletraffic Seminar, (2011): 176–181.

[16] Andersen, D., Theoretical approaches to node assignment., Unpublished Manuscript, `http://www.cs.cmu.edu/dga/papers/andersen-assign.ps`, 2002.

[17] Zhu, Y. and Ammar, M., Algorithms for assigning substrate network resources to virtual network components. Proceedings of IEEE INFOCOM'06, April 2006.

[18] Ma, Q. and Steenkiste, P., On path selection for traffic with bandwidth guarantees. Proceedings of IEEE ICNP'07, (1997): 191.

[19] Lu, J. and Turner, J., Efficient mapping of virtual networks onto a shared substrate. Tech. Rep. WUCSE-2006-35, Washington University in St. Louis, 2006.

[20] Yu, M., Yi, Y., Rexford, J., and Chiang, M. Rethinking virtual network embedding: substrate support for path splitting and migration. ACM SIGCOMM Computer Communication Review 38 , 2 (2008): 17–29.

[21] Eppstein, D. Finding the k shortest paths. SIAM Journal on Computing 28 , 2 (1998): 652–673.

[22] Chowdhury, N., Rahman, M., and Boutaba, R., Virtual network embedding with coordinated node and link mapping. Proceedings of IEEE INFOCOM'09, April 2009.

[23] Pióro, M. and Medhi, D. Routing, Flow, and Capacity Design in Communication and Computer Networks. Morgan Kaufmann Publishers Inc. San Francisco, CA, USA, 2004.

[24] Chowdhury, M., Samuel, F., and Boutaba, R., PolyViNE: policy-based virtual network embedding across multiple domains. Proceedings of the second ACM SIGCOMM workshop on virtualized infrastructure systems and architectures, ACM, (2010): 49–56.

[25] Houidi, I., Louati, W., Ameur, W., and Zeghlache, D. Virtual network provisioning across multiple substrate networks. Computer Networks 55 , 4 (2010): 1011–1023.

[26] Fajjari, I., Aitsaadi, N., Pujolle, G., and Zimmermann, H., VNE-AC: Virtual network embedding algorithm based on ant colony metaheuristic. Proceedings of IEEE ICC'11, IEEE, (2011): 1–6.

[27] Wong, E., Chan, A., and Yum, T. A taxonomy of rerouting in circuit-switched networks. IEEE Communications Magazine 37 , 11 (1999): 116–122.

[28] He, J., Zhang-Shen, R., Li, Y., Lee, C., Rexford, J., and Chiang, M., Davinci: Dynamically adaptive virtual networks for a customized internet. Proceedings of ACM CoNEXT'08, ACM, (2008): 15.

[29] Boyd, S. and Vandenberghe, L. Convex optimization. Cambridge university press, 2004.

[30] Chiang, M., Low, S., Calderbank, A., and Doyle, J. Layering as optimization decomposition: A mathematical theory of network architectures. Proceedings of the IEEE 95 , 1 (2007): 255–312.

[31] Palomar, D. and Chiang, M. A tutorial on decomposition methods for network utility maximization. IEEE Journal on Selected Areas in Communications 24 , 8 (2006): 1439–1451.

[32] Zhou, Y., Li, Y., Sun, G., Jin, D., Su, L., and Zeng, L., Game theory based bandwidth allocation scheme for network virtualization. Proceedings of IEEE GLOBECOM'10, IEEE, (2010): 1–5.

[33] Wang, C., Wang, C., and Yuan, Y., Game based dynamical bandwidth allocation model for virtual networks. Proceedings of IEEE ICISE'09, IEEE.

[34] Amazon Virtual Private Cloud (Amazon VPC)., http://aws.amazon.com/vpc/.

[35] Bertsekas, D. Nonlinear programming. Athena Scientific, 1999.

[36] Ousterhout, K. and Rexford, J., REEF: A REactivate, Efficient, and Flexible System for Internet Traffic Management. Master's thesis Princeton, 2011.

[37] CVXOPT: A python software for convex optimization., Available from: http://abel.ee.ucla.edu/cvxopt/.

[38] SCAPY: An interactive packet manipulation program., Available from: http://www.secdev.org/projects/scapy/.

[39] TCPREPLAY: A software to replay network traffic., Available from: http://tcpreplay.synfin.net/.

[40] TCPDUMP: A command-line packet analyzer., http://www.tcpdump.org/.

[41] Gao, X., Yu, H., Anand, V., Sun, G., and Di, H., A new algorithm with coordinated node and link mapping for virtual network embedding based on LP relaxation. Asia Communications and Photonics Conference and Exhibition, International Society for Optics and Photonics, (2010): 79881Y–79881Y.

[42] Javed, U., Suchara, M., He, J., and Rexford, J., Multipath protocol for delay-sensitive traffic. Proceeding of IEEE COMSNETS'09, IEEE, (2009): 1–8.

[43] Suksomboon, K., Satayapiwat, P., and Aswakul, C., Software development for automated network design supporting unicast and multicast traffics in next generation network. Proceedings of the 1st international conference on Simulation tools and techniques for communications, networks and systems & workshops, ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), (2008): 1–8.

[44] Cisco Data Center Infrastructure 2.5 Design Guide ., http://www.cisco.com/univercd/cc/td/doc/solution/dcidg21.pdf.

[45] Keller, E., Drutskoy, D., Szefer, J., and Rexford, J., Cloud Resident Data Center. Tech. Rep. TR-914-11, Princeton University, 2011.

[46] Al-Fares, M., Radhakrishnan, S., Raghavan, B., Huang, N., and Vahdat, A., Hedera: Dynamic flow scheduling for data center networks. Proceedings of the 7th USENIX conference on Networked systems design and implementation, USENIX Association, (2010): 19–19.

[47] Alizadeh, M., Greenberg, A., Maltz, D., Padhye, J., Patel, P., Prabhakar, B., Sengupta, S., and Sridharan, M., Data center tcp (dctcp). ACM SIGCOMM Computer Communication Review 40 , 4 (2010): 63–74.

[48] Abilene Backbone., Available from: http://abilene.internet2.edu/.

[49] Low, S. A duality model of TCP and queue management algorithms. Networking, IEEE/ACM Transactions on Networkings 11 , 4 (2003): 525–536.

[50] Kelly, F., Maulloo, A., and Tan, D. Rate control for communication networks: shadow prices, proportional fairness and stability. Journal of the Operational Research society 49 , 3 (1998): 237–252.

[51] Le Boudec, J., Rate adaptation, congestion control and fairness: A tutorial., http://moodle.epfl.ch/file.php/523/CC_Tutorial/cc.pdf, November 2005.

[52] Kleinrock, L. Queuing Systems (Vol. I). Wiley & Sons, NY, 1976.

[53] Fortz, B. and Thorup, M. Optimizing OSPF/IS-IS weights in a changing world. IEEE Journal on Selected Areas in Communications 20 , 4 (2002): 756–767.

Appendix A

Virtual network centralized algorithms

In this appendix, we give the detailed explanations for the virtual-network centralized algorithm used in **Chapter** IV. The throughput-sensitive traffic control problem is presented in Section A.1. Here, we prove the throughput-sensitive traffic control is convex problem and proportional fair. Section A.2 discusses delay-sensitive traffic control in which we present the piece wise linear approximation method to convert this problem to be solvable problem.

A.1 Throughput-sensitive traffic control

The throughput-sensitive traffic control for one virtual network of (4.1) can be formulated as an optimization problem follows

$$\begin{aligned} \text{maximize} \quad & \sum_i w_i \log\left(\sum_j z_j^i\right) - q\sum_{l\in L_2}\exp\left(u_l\right) & \\ \text{subject to} \quad & \sum_i\sum_j H_{lj}^i z_j^i \le y_l & \forall l \in L \\ & z_j^i \ge 0 & \forall i,j \\ \text{variables} \quad & z_j^i, & \end{aligned} \tag{1}$$

where w_i is the weight for demand i, z_j^i is the max flow rate available on path j for demand i. The link-path routing matrix with its column representing each path in source-destination pairs and its row representing each link is denoted by H_{lj}^i, where $H_{lj}^i = 1$ if demand i on path j uses link l and $H_{lj}^{i(k)} = 0$ otherwise. The capacity and utilization of link l are denoted as y_l and u_l, correspondingly. The importance of the low link congestion $\sum_{l\in L_2}\exp\left(u_l\right)$ in the objective function is denoted as q. If q is high, then the optimization will try to minimize the link utilization. If q is low, then the optimization will try to maximize throughput

Under the constraints of virtual-link capacity, the optimization problem (1) maximizes the traffic flowing through the virtual network, meanwhile prevents the virtual network's links from being congested. This optimization problem is convex and proportionally fair.

Because of the affine constraints, the convex property of (1) can be proven by showing its objective function is concave [29]. This objective function is composed by two basic functions $f(z^i) = \log(\sum_j z^i_j), \forall i$, where z^i is an allocated traffic vector for demand i via multi-path $\{j|\forall j\}$ and $g(u_l) = \sum_{l\in L_2} \exp(u_l)$. The following part proves the convexity of these two basic functions, and applies the composition rule [29] to justify the convexity of the objective function of (1).

The first and second derivative of $f(z^i)$ with z^i_j are calculated as follows

$$\frac{\partial}{\partial z^i_j} f(z^i) = \frac{1}{z^i_j} \qquad \forall i \tag{2}$$

$$\frac{\partial^2}{\partial z^i_j \partial z^i_p} f(z^i) = \begin{cases} \frac{-1}{{z^i_j}^2} & \text{if } j = p \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

From (3), Hessian $\nabla^2 f(z^i)$ is negative semi-definite, hence $f(z^i)$ is concave function with variable z^i. Let $g(u_l) = \sum_{l\in L_2} \exp(u_l)$. The first and second derivative of $g(u_l)$ is as follows:

$$\begin{aligned} \frac{\partial}{\partial u_l} g(u_l) &= \exp(u_l) \\ \frac{\partial}{\partial u_l \partial u_p} g(u_l) &= \begin{cases} \exp(u_l) & \text{if } l = p \\ 0 & \text{otherwise} \end{cases} \end{aligned} \tag{4}$$

The negative semi-definite of Hessian matrix $\nabla^2 g(u_l)$ implies $g(u_l)$ convex [29]. Hence, $g(z^i_j)$ is convex with z^i_j because it is the composition of a convex function $g(u_l)$ with a non-decreased affine function $u_l(z_j) = \frac{\sum_i \sum_j H^i_{lj} z^i_j}{y_l}$. The objective function of 1 is a composition of the two concave functions $f(z^i)$ and $-qg(z^i_j)$, then it's concave [29]. The constraints $\sum_i \sum_j H^i_{lj} z^i_j \quad \forall l \in L$ are affine with the variable z^i. Hence (1) is convex optimization problem [29].

Furthermore, the solution for this optimization problem is proportionally fair [50, 51]. A point is proportionally fair if it satisfies the following equation:

$$\sum_i \frac{z^i - z^{i*}}{z^{i*}} \leq 0 \tag{5}$$

The objective function in problem (1) without the second term can be presented as the function of traffic demand as

$$\begin{aligned} f(z) &= \sum_i w^i \log\left(z^i\right) \\ \frac{\partial}{\partial z^i} f(z) &= \frac{w^i}{z^i} \end{aligned} \tag{6}$$

Applying Taylor series to first order approximate $f(z)$ at the optimal point z^* as

$$\begin{aligned} f(z) &= f(z^*) + (z - z^*)^T \nabla f(z^*) + o(\delta) \\ f(z) - f(z^*) &= \sum_i \frac{z^i - z^{i*}}{z^{i*}} \end{aligned} \tag{7}$$

But $f(z)$ is a convex function, $f(z) - f(z^*) \leq 0$. And we have:

$$\sum_i \frac{z^i - z^{i*}}{z^{i*}} \leq 0 \tag{8}$$

The equation (8) proves that the optimal solution of (1) is also proportionally fair.

A.2 Delay-sensitive traffic control

The second example for the virtual network centralized algorithm, (4.1) is as follows:

$$\begin{aligned} &\text{minimize} \sum_i \sum_j z_j^i \sum_l H_{lj}^i (f_l + f(u_l)) \\ &\text{subject to} \sum_i \sum_j H_{lj}^i z_j^i \leq y_l \qquad \forall l \in L \\ &\qquad z_j^i \geq 0 \qquad \forall i, j \\ &\text{variables } z_j^i \end{aligned} \tag{9}$$

where z_j^i is max flow rate available on path j for demand i. The link path routing matrix with its column representing each path in source-destination and its row representing each link index is denoted by H_{lj}^i, where $H_{lj}^i = 1$ if demand $i$on path j uses link l and $H_{lj}^i = 0$ otherwise. The propagation delay is denoted by f_l. The queue in each node is modelled as M/M/1 and the total delay for a packet pass through a node is denoted by $f(u_l)$ and is calculate as follows [52]

$$f(u_l) = \frac{1}{c_l - b_l} \tag{10}$$

where c_l is capacity of link l, and b_l is load on link l. Let $U^{(1)}$ be denoted as the objective function of (9) and the objective function with the queuing delay from (10) is expressible as

$$\begin{aligned} U^{(1)} &= \sum_i \sum_j z_j^i \sum_l H_{lj}^i (f_l + f(u_l)) \\ &= \sum_l b_l (f_l + \frac{1}{c_l - b_l}) \end{aligned} \tag{11}$$

$$\tag{12}$$

The objective function in (11) is convex but it is difficult to solve. We use piece wise linear approximation as in [53] to linearly approximate the (11) as the following:

$$\max_{\forall b_l} U_l = \begin{cases} b_l f_l + \frac{1.5}{y_l} b_l, & 0 \leq b_l \leq \frac{y_l}{3} \\ b_l f_l + \frac{4.5}{y_l} b_l - 1, & \frac{y_l}{3} \leq b_l \leq \frac{2y_l}{3} \\ b_l f_l + \frac{30}{y_l} b_l - 18, & \frac{2y_l}{3} \leq b_l \leq \frac{9y_l}{10} \\ b_l f_l + \frac{70}{y_l} b_l - 54, & \frac{9y_l}{10} \leq b_l \leq c_l \\ b_l f_l + \frac{500}{y_l} b_l - 484, & y_l \leq b_l \leq \frac{11c_l}{10} \\ b_l f_l + \frac{5000}{y_l} b_l - 5434, & \frac{11y_l}{10} \leq b_l \leq \infty \end{cases} \tag{13}$$

Appendix B

List of notations

$1-\delta$	$\triangleq$	Probability of getting full avaiablility
$1-\epsilon$	$\triangleq$	Probability of getting limited avaiablility
γ	$\triangleq$	Reduction factor which is the ratio of capacity provided to a limited availability user comparing with the capacity provided to a full availability user
α	$\triangleq$	Activity level
a	$\triangleq$	Peak rate
N	$\triangleq$	Number of users in an aggregated demand
r^{full}	$\triangleq$	The minimum capacity for providing full availability
r^{lim}	$\triangleq$	The minimum capacity for providing limited availability
Q^n	$\triangleq$	The bandwidth demand of n^{th} user
N_{excl}^{full}	$\triangleq$	Number of users getting exclusive service with the capacity enough for providing full availability service to N^{full} users
N_{excl}^{lim}	$\triangleq$	Number of users getting exclusive service with the capacity enough for providing limited availability service to N^{lim} users
V^s	$\triangleq$	Set of all substrate nodes
E^s	$\triangleq$	Set of all substrate links
l	$\triangleq$	The general link index number
L	$\triangleq$	Set of all substrate network's link index number
k	$\triangleq$	Virtual network index number
K	$\triangleq$	Set of all virtual network's index number
K_l	$\triangleq$	Set of all virtual network's index number which has a link hosting on link l
L_k	$\triangleq$	Set of all substrate links hosted by virtual network k

i $\triangleq$ The traffic demand's index

j $\triangleq$ The path index for each traffic demand i

$H_{lj}^{i(k)}$ $\triangleq$ The link-path routing matrix for virtual network k with its column representing each path in source-destination pairs and its row representing each substrate link index which $H_{lj}^{i(k)} = 1$ if demand i of virtual network k on path j uses virtual network link l and $H_{lj}^{i(k)} = 0$ otherwise

$r_i^{(k)}$ $\triangleq$ Demand i of virtual network k

p_l $\triangleq$ price for one unit of traffic flowing through link l

$z_j^{i(k)}$ $\triangleq$ Max flow rate available on path j for demand i of virtual network k.

$z_j^{*i(k)}$ $\triangleq$ Optimal flow rate available on path j for demand i of virtual network k

$x_i^{(k)}$ $\triangleq$ An indication variable which has the value of 1 if demand i of virtual network k goes through path j, and value of 0 otherwise.

$y_l^{(k)}$ $\triangleq$ Bandwidth allocated to link l o virtual network k

$\tilde{y}_l^{(k)}$ $\triangleq$ A feasible bandwidth allocated to link l o virtual network k

$y_l^{*(k)}$ $\triangleq$ Optimal bandwidth allocated to link l o virtual network k

c_l $\triangleq$ Capacity of substrate's link l

$V^{(k)}$ $\triangleq$ Set of all virtual nodes in virtual network k

$E^{(k)}$ $\triangleq$ Set of all virtual links in virtual network k

m $\triangleq$ Number of paths for each traffic demand

$U^{(k)}()$ $\triangleq$ Utility function of virtual network k

$U^{(1)}$ $\triangleq$ Delay-sensitive utility function

$U_l^{(1)}$ $\triangleq$ Utility got from link l

u_l $\triangleq$ Utilization of link l

$w^{(k)}$ $\triangleq$ Weight of virtual network k

w_i $\triangleq$ Weight of demand i

$\lambda_l^{(k)}$ $\triangleq$ Dual variable of link l in virtual network k for the optimization sub-problem (4.1)

$\lambda_l^{*(k)}$ $\triangleq$ Optimal dual variable of link l in virtual network k

T	$\triangleq$	The period for substrate network to update virtual link bandwidth
β	$\triangleq$	Step size in gradient projection method to search for feasible optimal point
q	$\triangleq$	Priority for link low utilization
f_l	$\triangleq$	Propagation delay of link l
$f(u_l)$	$\triangleq$	Queuing delay of link l with utilization u_l
μ_l	$\triangleq$	Dual variable for the optimization master problem
p^*	$\triangleq$	Optimal solution for the substrate primal problem of (4.2)
$p^{*(k)}$	$\triangleq$	Optimal solution for the primal problem of
b_l	$\triangleq$	Load on link l

Appendix C

List of publications

Tri Trinh, Hiroshi Esaki and Chaodit Aswakul

"An analysis of careful overbooking for dynamically adaptive virtual networks", appeared in Proceedings of IEICE Technical Committee Workshop on Internet Architecture, 28-29 October 2010, Bangkok, Thailand, 2010. Content taken from **Chapter** III.

"Quality of Service Using Careful Overbooking for Optimal Virtual Network Resource Allocation", appeared in Proceedings of 8th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTI-CON 2011), 17-19 May 2011. Khon Kaen, Thailand, 2011. Content taken from **Chapter** III.

"Dynamic Virtual Network Allocation for OpenFlow Based Cloud Resident Data Center", appeared in IEICE TRANSACTIONS on Communications Vol.E96-B, No.01, pp.56-64, Jan. 2013. Content taken from **Chapter** IV.

Biography

Mr Trinh Minh Tri was born in 1981 in Hanoi, Vietnam. He received the Bachelor of Electronics and Telecommunication from Hanoi University of Science and Technology, Hanoi, Vietnam, in 2004, and the Master of Science in Electronics and Telecommunications from the same University, in 2006. He has been pursuing the Doctoral degree in Electrical Engineering at Chulalongkorn University, Bangkok, Thailand, since 2009. His research interests include Network Virtualization, Optimal Traffic Control for Cloud Network.