ABSTRACT

GAO, LINGNAN. Resource Allocation in Virtual Network Environments. (Under the direction of Dr. George Rouskas.)

Network Virtualization is one of the technologies with the potential to reshape the Internet architecture and introduce diversity into the current networks. Thanks to its perceived ability to overcome Internet ossification problem, a situation where innovation to the current Internet is almost impossible to achieve, network virtualization is seen as one of the most promising solutions to the next generation networks.

With the help of virtualization technologies, network services can be decoupled from the underlying infrastructures. Moreover, virtualization makes it possible for different network architectures and services to cohabit on the same infrastructure. In this context, how to allocate the resource from the physical networks remains a challenge. In this work, we address the problem on how to effectively allocate resources in the virtual network environment, which is crucial for the network performance.

As the first step of our work, we address the virtual network request partitioning problem. We consider a network-aware partition where the objective is to partition a network request while minimizing the traffic crossing different clusters. We address this problem as a clustering problem in eigenspace, whose Euclidean distance reflects the traffic intensity. Hence, the clustering result implies a partitioning of the virtual network request that minimizes inter-domain traffic.

Because network request may not remain static and will evolve over time, as a next step, we address the virtual request reconfiguration problem. To solve the reconfiguration problem, we presented an algorithm that will help to achieve load balancing by migrating part of the virtual nodes and virtual links.

In a virtual network environment, another crucial problem is the Network Function Virtualization, where the service request is the user traffic that needs to be processed by a series of virtual network functionalities, such as firewalls, load balancer, in a given order. One of the key challenges in realizing Network Function Virtualization is the service chain routing problem where the traffic must traverse various components instantiated on the underlying network to fulfill the service goal. We consider the service chain routing problem with the goal of minimizing the maximum network congestion in an online fashion. To this end, we present a simple yet effective online algorithm in which the

routing decision is irrevocably made without prior knowledge of future requests.

We carried out evaluation for the above resource allocation strategies, and numerical results are presented to validate the effectiveness of our approaches.

CONTENTS

LIST OF FIGURES

Chapter 1

Introduction

1.1 Virtualization for the Networks

The Internet had achieved a great success in the past few decades. With billions of users and numerous services supported at the application layer, today's Internet is truly reshaping and inter-connecting the world. Despite the fast growth both in size and the diversity of applications it supports, there has been significant slowing down when it comes to bringing in innovation to the Internet itself, as new architectures, protocols, and services are becoming increasingly difficult to integrate into today's network. One example to this slow evolution is exemplified by the time and difficulties it takes to transit from IPv4 to IPv6, a process that started two decades ago [56], and remains far from completion.

The problem above is known as the Internet ossification problem, with multiple contributing factors. First, the dedicated and proprietary nature of networking devices requires non-trivial time and effort to make even small changes. Second, any change to the existing network architecture would require consensus among all the Internet Service Providers (ISPs), making the implementation of a new architecture or protocol hard to realize. Moreover, the scale of today's Internet adds more difficulties to it. As a result, nearly all the changes to the network has been achieved via the incremental deployment [2].

With the help of the virtualization technologies, much of the above-mentioned problem can be mitigated, if not eliminated. The decoupling the networking services with its underlying hardware introduces the more flexibility to the networks. The same set of

physical infrastructure can embrace a diversity of network architectures, protocols, and services. Through the abstraction of the underlying infrastructure, it simplifies the effort to develop, deploy and maintain new services, thus lower the overall operational cost.

A number of networking architectures centering on virtualization have been proposed in this regards. Network Virtualization and Network Function Virtualization are two of them. In the subsequent paragraph, we shall briefly discuss the background on the two network paradigms, and the resource management problems in Network Virtualization and Network Function Virtualization environment.

1.1.1 Network Virtualization

Network Virtualization is seen as the enabler for the future generation of the networks. Through the virtualization techniques, it allows heterogeneous network to cohabit on the same infrastructure. Originally, network virtualization was proposed as a means to build a platform to the testing and evaluation of innovative network architectures or protocols on top of a legacy network environment, but now it is considered as one of the most promising paradigms for the next generation networks [2,16,17]. With network virtualization, the traditional role of ISPs is decoupled into two logical entities: Service Provider (SP) and Infrastructure Provider (InP) as shown in Figure 1.1.

In this context, InPs are responsible for the provisioning and management of the infrastructure, and offer their available resources as a service to SPs. By aggregating resources from multiple InPs, a SP can then create and deploy virtual networks on top of the leased infrastructure, and then deliver network services to the end users. The service provided to the users will be in the form of virtual networks (VNs).

Such a separation brings a number of benefits to network architectures including but not limited to [7,16,17]:

- **Heterogeneity**: the same set of underlying physical infrastructure may support heterogeneous virtual networks, thus the services delivered to the users are no longer limited by the type of physical resources; in addition, the same VN may be mapped onto heterogeneous underlying physical infrastructures.

- **Flexibility and Manageability**: the topology of each virtual network that the SP creates is customizable. Thus, the topology of the virtual network, including the virtual routers and switches, may be tailored towards the needs of users. Moreover,

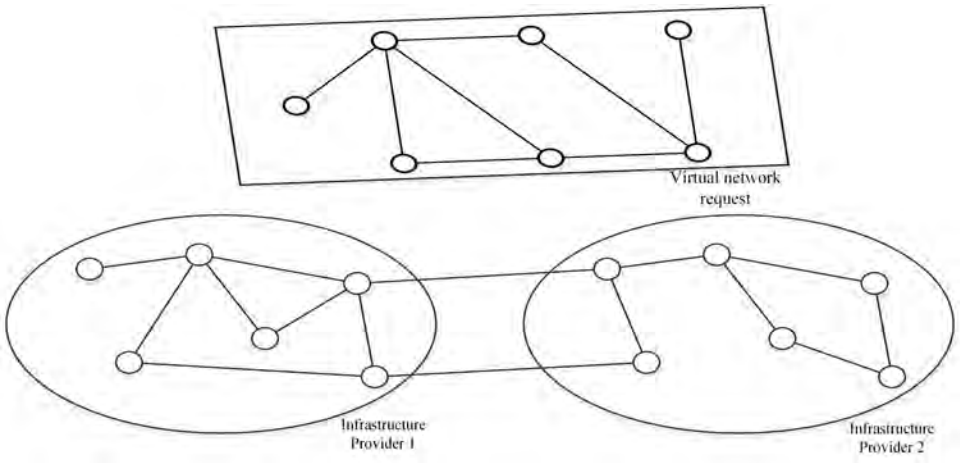

Figure 1.1: Virtual Network Environment [16]

as the virtualization provides an abstraction of the underlying physical resources, it simplifies the problem of network management.

- **Co-existence**: with virtualization, multiple users can co-exist on the shared infrastructure, and the service provider may decide how to pool the physical resources and how to aggregate requests so as to improve the overall utilization of the underlying infrastructure.

- **Security and Isolation**: the traffic of different users on a shared infrastructure can be isolated to the point where each user only has the view on its own slice of network. This ensures that traffic from one user will not interfere with that of another. Also, the isolation of traffic helps to prevent security issues.

1.1.2 Network Function Virtualization

Network function virtualization (NFV) [35] is yet another networking paradigm that harnesses the power of virtualization. Through the virtualization of the conventional middleboxes (e.g., firewall, load balancer), NFV eases the difficulties in the developing, deploying and maintaining the networking services in today's network.

The conventional way of the service provisioning relies on the deployment of proprietary hardware to support the network functions that are part of existing services. These dedicated equipment and hardware are not only difficult to maintain, but also

hard to extend their functionalities. This makes it difficult for the service provisioning to accommodate a diverse and sometimes short-lived the network service requests. All those factors result in a long production cycle and a high operational expenditure which impedes the agility of deploying new services.

With the help of virtualization techniques, NFV relies on the commercial off-the-shelf hardware to replace the existing networking devices [30]. The most basic element in the NFV ecosystem is the Virtual Network Function (VNF), representing a single functional block. The network function can be implemented as the software and runs on the general purpose server. Based on demand, the service chain combines one or more network functions to accomplish the service objective. For a conventional way of service provisioning, packets need to go through the NFs in a given order, to deliver an end-to-end network service. Same will apply to the NFV environment, where packets are routed through VNFs sequentially. The VNFs will process those packets so as to deliver the network service.

Such a paradigm opens the door for network operators to implement both existing and future networking functions as software modules and consolidate them on general-purpose commodity servers based on demand. This approach simplifies the deployment process and introduces flexibility in the service provisioning.

1.2 Resource Management Challenges

Despite the promising outlook of the virtualization technology that can bring, one challenge that both the network virtualization and network function virtualization face is the resource management problem, i.e., how to effectively allocate the available resource from the underlying infrastructure to the virtual requests. These problems are known as Virtual Network Embedding (VNE) and Network Function Virtualization Resource Allocation problems (NFV-RA) respectively. We shall discuss resource management problems in more details in the subsequent paragraphs.

1.2.1 Virtual Network Embedding Problem

The resource allocation problem for the network virtualization problem is referred as the VNE problem. Such a problem aims at find a mapping of the virtual request to the

underlying networks. Typically, the mapping of virtual network requests involves two sub-problems:

- **Virtual Node Mapping**, where a virtual node is instantiated on one physical node, and obtains the available resource from that substrate node accordingly
- **Virtual Link Mapping**, where the traffic of one virtual link is routed along a physical path, and the bandwidth is reserved accordingly. The endpoints of the physical path must be the substrate nodes where the virtual node is embedded upon.

Through the abstraction of the physical hardware, there are multiple ways to host one virtual network request. For example, for one virtual node, any substrate node with enough processing power can host that virtual node, while each virtual link may be along a path with enough bandwidth. This results in a large solution space for the potential mapping of the virtual network requests. However, it is to the interest of the network operator to obtain an efficient mapping of the virtual network requests.

Depend on the embedding objectives, an effective virtual network embedding strategy would help to deliver a better service to the end-users and to lower the operational cost. Those objectives under consideration may include enforcing QoS requirement [53], energy-efficiency [51] maintaining the survivability of the virtual network [50], load balancing [61] or maximizing revenue from existing infrastructure without service degradation [62].

The real challenge in solving virtual network embedding lies in the fact the VNE problem is NP-hard, i.e., it may not be possible to find an optimal solution within a reasonable amount of time. Even in the simplest case, where the network request remains static and is known in advance, this problem remains NP-hard [27]. How to design a resource allocation algorithm with a near-optimal performance becomes a critical issue.

In addition to different embedding objectives, there are additional aspects to consider: the support from the underlying infrastructure determines whether the traffic of a virtual link can route along single or multiple paths; the online nature of the virtual network request calls for dynamic embedding algorithm. All those embedding objectives and appliance scenarios diversify the embedding requirement and motivate the design of the various embedding algorithm to meet all those demands. A comprehensive survey on those embedding schemes can be found in [27].

1.2.2 Network Function Virtualization Resource Allocation

Unlike the service provisioning in a conventional network, where the network operator statically configures how the packets pass through the middle boxes, the service provisioning under NFV environment can exploit the flexibility of the virtualization techniques to dynamically coordinate the virtual network functions and form the service chain. However, how to allocate the resources from the underlying networks for Network Service (NS) requests also raise the issue of resource management.

The resource allocation problem can generally be depicted by two phases, the *Service Chain Composition* phase, and the *Service Chain Embedding* phase, as illustrated by the Fig. 1.2.

Within the NFV environment, some Network Services need to concatenate multiple VNFs to achieve the service goal. The NS request comes with the traffic demand and the specification on the necessary VNF types. Moreover, there may exist dependent relationships on the VNFs, meaning one type of the VNF needs to process a packet before the packet goes to another type of VNF. This procedure of dynamically concatenating multiple VNFs for the service deployment to meet the service goal is known as *Service Chain Composition* problems.

As the Service Chain Composition phase determines a service chain, taking it as an input, how to find an appropriate mapping of the service chain to the underlying infrastructure remains an issue. The process of finding an efficient mapping of a service chain to the underlying network is known as the *Service Chain Embedding* problems. Similar to the VNE problem, the Service Chain Embedding problem also involves the mapping of the virtual nodes and virtual links, and different embedding objectives and scenario would call for a different embedding scheme.

The two phases, Service Chain Composition and Service Chain Embedding, are both important to fulfill the goal of the service provisioning. The Service Chain Composition will strategically determine how the network service will be fulfilled and carried out, while Service Chain Embedding problem considers how to coordinate the physical resource to support the network service.

To deploy an NS, it is natural to decompose these two phases and tackle them respectively, meaning the network operator can compose a service chain based on the policy constraints and then map it to the physical infrastructure. For example, in the work of [46], the authors model the service chain composition for an NS as a context-free

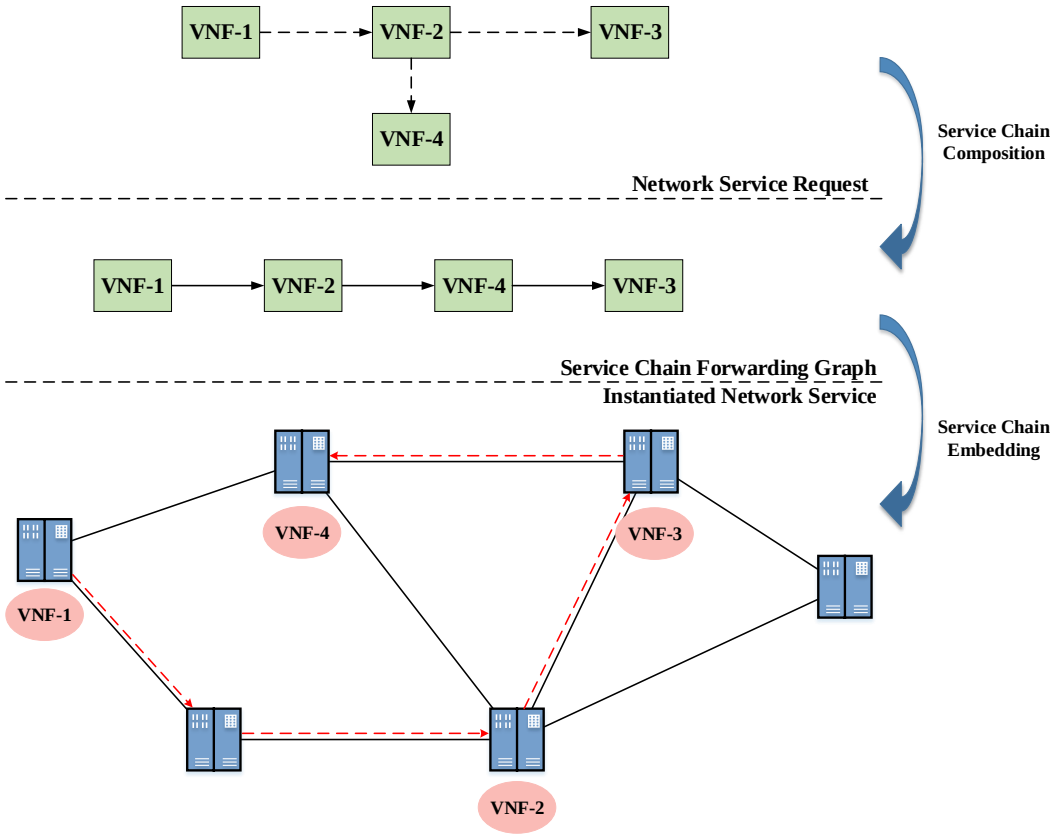

Figure 1.2: NFA Resource Allocation

grammar, and greedily select the service chain that will minimize the traffic requirement.

With a valid service chain in the form of the Service Chain Forwarding Graph, the Service Chain Embedding phase shall take the composed service chain as an input, and map it upon the physical network with a predefined service goal, such as end-to-end delay minimization [8] or provisioning cost minimization [42].

Although the resource allocation is carried out during the embedding phase, the Service Chain Composition nevertheless has a major impact on the resource management. It is possible that the same service chain may have multiple valid Service Chain ForwardingG Graph, and this, in turn, leads to different resource requirement. Therefore, there are efforts directed towards a coordinated service chain composition and embedding. For example, the work of [9] proposes an heuristic algorithm based on the Depth First Search to embed the Network Service to the underlying networks with the objec-

tive to increase the request acceptance ratio. The service chain forwarding graph will be adjusted accordingly during the embedding process.

1.2.3 VNE and VNF-RA

From the discussions above, we can see that the VNE and the VNF-RA problem bear remarkable resemblance: both involve mapping the requests onto the substrate network with the finite capacity to meet the predefined objectives, which typically involves the mapping of virtual nodes and virtual links, while the process itself is NP-hard.

However, the difference exists both in terms of the request and the support from the underlying physical infrastructure.

When it comes to request, the virtual network request can be represented by a weighted graph, with the edge weights stand for the bandwidth requirement and node weight for the processing power demand. This is not the case for the VNF-RA problem, whose request consists of the source, destination, traffic demand, required types of the VNFs, and dependencies among the VNFs.

In the context of VNE, unless there is a policy constraint, such as geographical consideration, the virtual node can be mapped on to any substrate node. The network operator generally does not need to consider the interactions between two virtual network requests. This is not the case for the NFV-RA. For the VNFs required by an NS request, it can either reuse the existing VNFs or instantiate new VNFs. Thus, for the Service Chain Embedding, the network operator has to decide whether or not to instantiate new VNFs to accommodate new service request. And to embed the service chain, each virtual function node can only be placed onto the matching VNF instances.

In short, the NFV-RA is more complicated as it involves the composition of the service chain, decision on whether to instantiate new VNFs or to reuse the existing ones. However, in the embedding phase, the topology of the Service Chain Forwarding Graph is simpler, as it may either be a path or a tree, while the topology for the virtual network may be arbitrary weighted graph.

1.3 Contribution

In this work, we focus on algorithm design for the resource allocation problems in a virtual network environment:

First, we consider the problem of how to partition a virtual network request in a multi-domain environment so as to minimize the inter-domain traffic. For the VNE problem, mapping virtual requests to multiple domains may be required for various reasons, including load balancing [58], managing the embedding cost [33], or policy requirements [20,47]. As inter-domain traffic is more expensive than intra-domain traffic, when we distribute a virtual network across two different domains, how to minimize the traffic across different domains becomes critical.

To this end, we consider the problem of virtual network request partitioning and present an algorithm inspired by spectral clustering to partition virtual network nodes under capacity constraints. Based on Laplacian transformation of the matrix, the pair of virtual nodes with intensive traffic between them tend to stay close to each other on eigenspace. This allows us to partition the virtual network on the eigenspace. The clustering is done using the constraint k-means, which clusters the virtual network request on eigenspace with respect to the capacity of that cluster. As a further optimization, we use a simulated annealing-based algorithm to further refine the partitioning.

Next, we handle the problem of how to re-embed the virtual network so as to adapt to the dynamic changes in the network. Since resource allocation problem is online in nature: the lifetime of the VN requests may be finite and arbitrary; the arrival pattern of VN requests may be unpredictable; and the resource demands for existing VNs may change whenever users decide to scale up or down their requirements. If these dynamics are not taken into account and resource allocation is viewed as a purely static problem, the whole infrastructure may drift into an inefficient configuration that results in degraded performance for existing VNs and a higher rejection rate for subsequent VN requests [62].

Our approach is to design an algorithm to reconfigure the VN embedding on a periodic basis. The mapping of virtual nodes and links to infrastructure resources will be updated in response to changes in VN requests. This is referred to as the virtual network reconfiguration (VNR) problem [22], and its objective is to improve resource utilization (for providers) and enhance performance (for users). Similar to the VNE problem, VNR is also concerned with the mapping of virtual nodes and links, but it also takes the existing configuration into account.

To this end, we first present a link-arc based mixed integer programming (MIP) formulation of the reconfiguration problem. Our objective of is to minimize the maximum utilization of substrate nodes and links while bounding the number of virtual nodes that

have to be migrated. This problem is NP-hard, since it reduces to the VNE problem if there is no bound on the number of virtual nodes to be migrated. Therefore, we decompose the problem into two phases, namely, virtual node selection and virtual node remapping. The first phase selects the virtual nodes to be migrated, while the second selects the new substrate nodes for the migrated virtual nodes. For the first phase, we solve the linear programming (LP) problem derived from the MIP. Specifically, we use a path-based formulation that achieves a near-optimal solution to the LP problem without intensive computation overhead. For the second phase, we use a Markov-chain based approach to select candid substrate nodes and solve a multi-commodity flow (MCF) problem for link reconfiguration.

Finally, we consider the problem of service chain routing problem, which is a sub-problem in the Service Chain Embedding phase for the NFV-RA problem. The objective of the service chain routing problem is to route the user traffic along a path that starts at the source node, passes through the network locations where VNFs are implemented, and finally reaches the destination node. In an offline scenario, where all service requests are known in advance, service chain routing is NP-hard; this result follows from the fact that the unsplittable flow problem, which was proven to be NP-hard in [5], is a special case of the service chain routing problem. In practice, the service chain routing is an online problem: service requests may arrive at arbitrary times and they must be placed onto the network without prior knowledge of future requests. These conditions pose additional challenges in developing effective and efficient algorithms for the online problem.

In our work, we consider the service chain routing with the goal on minimizing the congestion in the network. Taking the online nature of the service request into account, we present a simple yet efficient online algorithm to minimize the maximum utilization. The algorithm we develop can achieve an $O(logm)$ competitive ratio, in which m is the number of the edges in the substrate network. And we show that the competitive ratio is asymptotically optimal.

1.4 Structure of the Dissertation

The remainder of this dissertation is organized as follow:

Chapter 2 describes a virtual network partitioning strategy to partition the virtual network request across different domains in a way that the inter-cluster traffic is mini-

mized.

Chapter 3 presents a virtual network reconfiguration strategy that aims to achieve load-balancing while minimizing the cost relocating virtual nodes.

Chapter 4 presents an online service chain routing algorithm for the network function virtualization with the objective to minimize the maximum system utilization.

In Chapter 5, we summarize our work and propose future research directions.

Chapter 2

Virtual Network Request Partitioning

As discussed in the previous chapter, the objective of virtual network request partitioning problem is to partition the virtual network requests into a set of clusters so as to minimize the inter-cluster traffic. The partitioning should be done in a way that the capacity constraint of each cluster would be satisfied. In this chapter, we develop a clustering algorithm based on spectral clustering to handle this problem. Previous work related to this topic is reviewed in Chapter 2.1. In Chapter 2.2, we formally define the virtual request partitioning problem, and then present an algorithm based on spectral clustering to solve this problem in Chapter 2.3. Finally, in Chapter 2.4, we present numerical result to demonstrate the effectiveness of this algorithm.

2.1 Related Work in Virtual Network Request Partitioning

Several studies [33, 47, 58] have addressed the virtual request partitioning problem using max-flow, min-cut schemes. With existing algorithms, it is possible to compute efficiently the maximum flow between a pair of nodes and obtain the minimum cut between them. The work in [33] recursively uses the max-flow, min-cut approach to partition the network into the desired number of clusters. In [47, 58], a clustering approach based on Gomory-Hu trees is explored. A Gomory-Hu tree represents the $n-1$ minimum $s-t$ cuts in a

graph of n nodes. By removing the $k-1$ least weight edges of this tree, a partition of the n nodes into k clusters is obtained that is close to optimal. One shortcoming of this method is that the resulting clusters may be highly imbalanced in terms of the number of nodes they contain.

The virtual network embedding problem across multiple domains has been considered in [41], where it was proposed to use iterative local search (ILS) to partition the virtual request. For this problem, ILS starts with a random clustering, following which a sequence of solutions is generated by randomly remapping some of the nodes to other clusters. Of these solutions, the one that improves upon the current solution the most is kept, and the algorithm iterates until a stopping criteria is met. Despite the simplicity of this method, it is hard to guarantee the quality of the solution within a limited time. In a related study, a general procedure for resource allocation in distributed clouds was presented in [20]. The objective was to select the data centers, the racks, and processors with the minimum delay and communication costs, and then to partition the virtual nodes by mapping them onto the selected data center and processors.

In [13], a series of spectral partitioning algorithms are reviewed and summarized. Most of those spectral partitioning techniques use the median of the Fiedler vector, the second smallest eigenvalue of the Laplacian matrix, to partition the graph into two parts. Then, by recursively applying this method, one can partition the graph into $2^n,\ n \geq 1$, parts. These algorithms have two limitations. First, the node weights related to load demands are not taken into consideration; consequently, the load may not be balanced well across the various clusters. Second, the number of clusters is limited to powers of two. In [31], a spectral partitioning based method that makes use of multiple eigenvectors of the Laplacian matrix is proposed. While this work considers the node weight balancing and makes use of multiple eigenvectors to produce a solution with a high-quality, it only partitions a graph into two, four or eight parts.

In contrast to existing works, our algorithm exploits multiple eigenvectors towards a high-quality solution and allow network requests to be partitioned into arbitrary clusters. In addition, we make sure the capacity constraint of each cluster shall be satisfied.

2.2 Problem Statement

We model the communication between virtual nodes as a traffic matrix $TM = [t_{ab}]_{N \times N}$, where element t_{ab} represents the amount of traffic from virtual node a to b. Each virtual node is associated with a resource requirement Req_a, and each cluster k is associated with a capacity threshold Cap_k.

With these definitions, partitioning the set of virtual nodes into K clusters so as to minimize the inter-cluster traffic can be formulated as the following Integer Linear Programming (ILP) problem:

$$\textbf{minimize} \quad \sum_{k} \sum_{ab} t_{ab}(1 - y_k^{ab}) \tag{2.1}$$

$$\textbf{subject to} \quad \sum_{a} Req_a x_k^a \leq Cap_k, \qquad \forall h \tag{2.2}$$

$$x_k^a + x_k^b \leq y_k^{ab} + 1 \qquad \forall a, b, h \tag{2.3}$$

$$y_k^{ab} \leq x_k^a \qquad \forall i, j, h \tag{2.4}$$

$$\sum_{k} x_k^a = 1, \qquad \forall a \tag{2.5}$$

$$x_k^a = \{0, 1\}, y_k^{ab} = \{0, 1\} \tag{2.6}$$

The binary variable $x_k^a \in \{0, 1\}$ here indicates if virtual node a is assigned to cluster k while binary variable $y_k^{ab} \in \{0, 1\}$ indicates if virtual nodes a and b are both mapped onto cluster k.

Constraint (2.2) ensures that the amount of resources assigned to each cluster will not exceed its capacity limit. Constraint (2.3) and constraint (2.4) guarantees consistency between decision variable x and y. Constraint (2.5) makes sure that virtual node a is assigned to exactly one cluster. This formulation is equivalent to the (k, v)-balanced partitioning problem, which is an NP-complete problem [3].

For the solving of the virtual request partitioning problem, we apply a spectral clustering based [54] approach. A high level idea of our approach is illustrated in the Figure 2. We aim to partition the virtual request that consists of N nodes. Starting with a N-by-N traffic matrix TM, we compute the normalized Laplacian matrix L_{sym} of the given traffic matrix TM. We then compute the eigenvectors that associate with the first K smallest eigenvalues of matrix L_{sym}, and form a N-by-K matrix Z based on the k-smallest eigen-

Algorithm 1 Spectral clustering based Virtual Request Partitioning Algorithm

Input:

TM: $N \times N$ traffic matrix of virtual nodes
K: number of clusters
Req: resource requirement vector of virtual nodes
Cap: capacity vector of the clusters

Output:

The cluster to which each VNode belongs

1: Construct diagonal matrix D with $d_{aa} = \sum_{b=1}^{N} t_{ab}$
2: Compute unnormalized Laplacian $L_{sym} = D^{-1/2}(D - TM)D^{-1/2}$
3: Obtain the eigenvector associated with the K smallest eigenvalues of matrix L_{sym}
4: Let matrix U contain the above eigenvectors as columns
5: Let z_a be the vector associated with a^{th} row of U.
6: Cluster the points $(z_a)|_{a=1...N}$ under the capacity constraints Cap via **Constrained k-means**
7: Refine the partitioning result by **Greedy-Refinement** based Simulated Annealing

vectors. Denote a^{th} row of the matrix, a K dimensional vector, as z_a. The virtual node a will be associated with z_a. We can think of the z_a as the coordinates for the virtual node a on a K dimensional space, and the matrix Z as the collection of coordinates for the N data points. This K-dimensional space is referred as Eigenspace. With given property of the Laplacian matrix, the nodes that have higher traffic among them tend to stay closer on this Eigenspace. Thus, we can perform clustering for those data points $(z_a)_{a=1,...,N}$. Clustering will assign the nodes to different clusters so as to minimize the total distance between the nodes within each cluster on the Eigenspace, which implies this assignment maximizes the intra-cluster traffic and eventually minimize the inter-cluster traffic. In order to satisfy the capacity constraints, we perform constrained k-means algorithm for clustering. Upon the completion of the clustering, we obtain our initial clustering results. The result can be further refined using the Greedy-Refinement algorithm based on the traffic matrix TM. It will iteratively assign the Virtual Node to another cluster if such an assignment would lower the inter-cluster traffic. And to achieve a better partitioning result, we can combine the Greedy Refinement algorithm with Simulated Annealing. The whole procedure is outlined as Algorithm 1, and is explained in detail in the remaining of this chapter.

2.3 Spectral Clustering Based Network Request Partitioning Algorithm

Spectral clustering [54] is used to find a set of clusters such that the edges between clusters have low weights (in this case, the weights would represent the pairwise traffic). An important feature of spectral clustering is that, unlike the conventional min-max flow approach, it can avoid the creation of imbalanced partitions whereby some clusters are assigned a much larger number of nodes than others.

2.3.1 EigenSpace

Given a $N \times N$ traffic matrix TM, the normalized Laplacian matrix is defined as $L_{sym} = D^{-1/2}(D-T)D^{-1/2}$, where D is a diagonal matrix with element $d_{aa} = \sum_b t_{ab}$.

Let $P_1, \ldots, P_K$ be a partition of the set of N virtual nodes into K sets (clusters), i.e., the sets P_k are pairwise disjoint and their union is $\{1, \ldots, N\}$. Further, let $\bar{P}_k$ be the complement of set P_k. We define the $NCut$ metric as:

$$NCut(P_1, P_2, ..., P_K) = \sum_{k=1}^{K} \frac{cut(P_k, \bar{P}_k)}{vol(P_k)} \tag{2.7}$$

where the numerator represents inter-cluster traffic (i.e., between nodes in P_k and nodes not in P_k) and the denominator denotes total traffic within cluster P_k.

Minimizing $NCut$ will result in a set of K clusters that have low inter-cluster traffic, while the presence of $vol(P_k)$ in the denominator will prevent the creation of clusters with few nodes, and hence, cluster sizes will not be highly imbalanced. The normalized Laplacian has the following property that allows us to find an approximate solution to the $NCut$ problem efficiently: for a given N-by-k matrix H, if we take h_{ab} as:

$$h_{ab} = \begin{cases} 1/\sqrt{vol(P_k)} & if\ v_i \in P_j \\ 0 & otherwise \end{cases} \tag{2.8}$$

the following equation would hold [54]: $Tr(H^T L H) = \sum_{i=1}^{N} \frac{cut(P_k, \bar{P}_k)}{|vol(P_k)|} = NCut(P_1, P_2, ..., P_k)$.

Also observe that $H^T D H = I$. Therefore, we can reformulate the problem of mini-

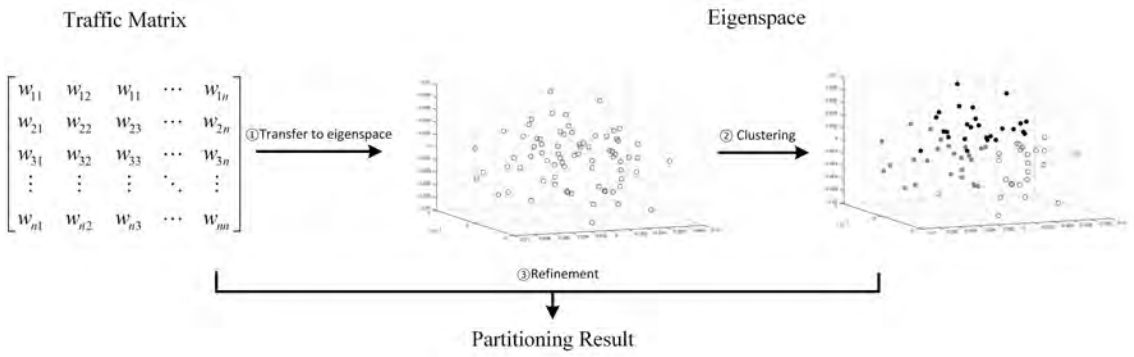

Figure 2.1: General Procedure for Virtual Request Partitioning

mizing $NCut$ as follows:

$$\begin{aligned} \textbf{minimize} \quad & Tr(H^T L H) \\ \textbf{subject to} \quad & H^T D H = I \\ & h_{ab} = \{1/\sqrt{vol(P_k)}, 0\} \end{aligned} \tag{2.9}$$

We can obtain an approximate solution to this problem in polynomial time by relaxing the last condition and taking $Z = D^{-1/2}H$. The problem then becomes:

$$\begin{aligned} \textbf{minimize} \quad & Trace(Z^T L_{sym} Z) \\ \textbf{subject to} \quad & Z^T Z = I \end{aligned} \tag{2.10}$$

According to the Rayleigh-Ritz Theorem, the solution to problem 2.10 would be to take Z as the k smallest eigenvectors of L_{sym}, i.e. the eigenvectors corresponding to the k smallest eigenvalues.

Let matrix Z be a N-by-k matrix that contains the above k eigenvectors as columns, and let z_i be the vector associated with the a-th row of Z, and we take z_i as coordinates for the virtual nodes on the Eigenspace. The data points will stay close to each other in the Eigenspace if the traffic between them is high, and this will allow us to obtain the final solution using clustering algorithm. A formal proof of this property that established in the matrix perturbation theory is presented in [49].

To obtain the final solution, clustering will be performed to cluster the points $(z_a)_{a=1,\ldots,N}$ while satisfying the capacity constraints Cap.

2.3.2 Constrained K-means

Conventional spectral clustering uses the k-means algorithm [45] to cluster the data points z_a. One drawback of the k-means algorithm is that it may converge to a solution in which some clusters have very few data points while others are overloaded. Therefore, we use the constrained k-means algorithm proposed in [12]. Given a set of data points, the constrained k-means algorithm aims to find a set of cluster centers $C_1, C_2, \ldots, C_K$, such that the sum of distances between each node and the center it is assigned to is minimal. Specifically, the problem solved in [12] is:

$$\begin{aligned} \textbf{minimize} \quad & \sum_{a=1}^{N}\sum_{k=1}^{K} x_k^a \cdot \left(\frac{1}{2}\|z_a - C_k\|_2^2\right) \\ \textbf{subject to} \quad & \sum_{k=1}^{N} x_k^a = 1,\ \forall i; \quad x_k^a \geq 0,\ \forall i,\ \forall h. \\ & \sum_{i} x_k^a \geq \tau_k,\ \forall k \end{aligned} \tag{2.11}$$

In this formulation, x_k^a is a selection variable denoting whether data point a belongs to cluster k. The last constraint is used to control the size of each cluster, i.e., to ensure that each cluster has size at least equal to τ_k.

In the virtual request partitioning problem, the resource requirement for each cluster should not exceed its capacity. To this end, we replace the constraint $\sum_i x_k^a \geq \tau_k$ with $\sum_i Req_a x_k^a \leq Cap_k$, and follow the iterative method proposed in [12].

Given N data points $z_1, z_2, ..., z_N$, K cluster center points $C_1^t, C_2^t, ..., C_K^t$ at iteration t, and capacity limit Cap_k for cluster k, the cluster center for iteration $t+1$ is computed using the following steps.

Cluster Assignment. Given the fixed cluster center points C_k, find the selection variables so that the distance between the data points and the corresponding cluster center is minimized.

$$\begin{aligned}
\textbf{minimize} \quad & \sum_{a=1}^{N}\sum_{k=1}^{K} x_k^a \cdot \left(\frac{1}{2}\|z_a - C_k\|_2^2\right) & & \\
\textbf{subject to} \quad & \sum_{k=1}^{K} x_k^a = 1, & \forall a & \qquad (2.12)\\
& \sum_{a} Req_a x_k^a \leq Cap_k, & \forall k & \\
& x_k^a \geq 0, & \forall a, \forall k &
\end{aligned}$$

Cluster Update. Compute the center point at iteration $t+1$ as:

$$C_k^{t+1} = \begin{cases} \frac{\sum_{i=1}^{N} x_k^a(t) z_a}{\sum_{i=1}^{N} x_k^a(t)} & if \sum_{i=1}^{N} x_k^a(t) > 0 \\ C_k^t & otherwise \end{cases} \qquad (2.13)$$

It was shown in [11] that cluster assignment is equivalent to the Minimal Cost Flow problem. We now show that this cluster assignment subproblem can be solved optimally within $O(KN \log N)$ time using a greedy approach.

We first reduce cluster assignment to the Min-Cost-Flow problem following the steps outlined in [11]. The supply from the source node (src) and the demand by sink node (dst) is equivalent to the total requirement $\sum_{a=1}^{N} Req_a$. src is connected to all the data points $(z_a)|_{a=1,\ldots,N}$, and each data point is connected with all the cluster centers, while cluster centers are connected to dst. Each edge is associated with a weight tuple $(Price, MaximumCapacity, Flow)$. The $Price$ from data points to cluster centers are set to the corresponding distance, while on other edges it is set to zero. The $MaximumCapacity$ from src to the data points is the resource requirement Req_a and from cluster center k to dst is Cap_k; on other edges, it is set to infinity. An example of the representation of a cluster assignment problem to an Min-Cost-Flow network is shown in Figure 2.2 (a).

Now, we show how this problem can be solved with a greedy approach. First, ascending sort the price on all the paths from src to dst and augment flow accordingly. When we think of this problem in terms of negative cycle canceling algorithm, each time we augment the flow by f on a path, we create a reverse path with negative price on the

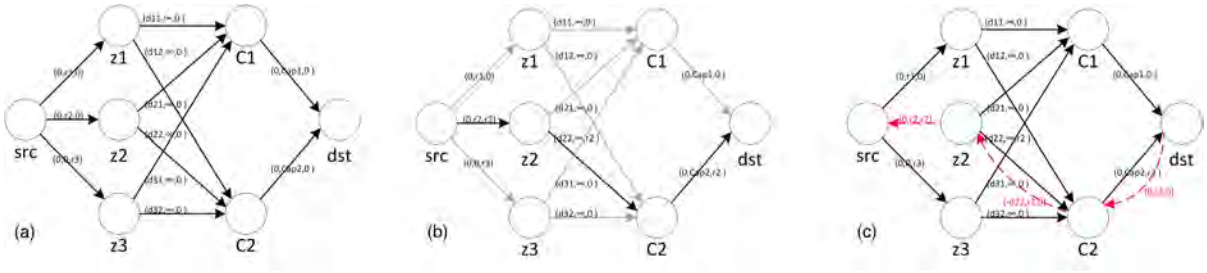

Figure 2.2: Min-Cost-Flow view of clustering assignment subproblem

residual graph. An example can be found in Figure 2.2 (b) and (c).

A brief proof of optimality is as follows. Denote the iteration to augment flow on path i as t_i. At t_i, we augment flow on path i. For $t_j > t_i$, no negative cycle will form on the residual graph involving the reversed path i, because the price of path j will be no less than of path i. Also, for path j with $t_i > t_j$, the price of path i is higher, hence a negative cycle will be formed only when we take forward direction from on path j and backward on i, which is impossible. This is from the fact on path j, the capacity on src to a data point or from cluster center ot dst is depleted. In the former case, we cannot find a forward path from src to the data point, so path j is blocked, and no cycle will form. The same applies to the latter case when path i and j go through a different cluster center. If they pass through same cluster center, then, we cannot augment the flow on path i, because there is no available capacity from the cluster center to dst, so no negative circle will form. In conclusion, no negative cycle can be found and the solution will be optimal.

At each step, denote the remaining capacity of cluster k as Rm^h_{cap}, and the remaining resource requirement of each virtual node a as Rm^i_{res}. Our constrained-k-means with greedy cluster assignment algorithm is shown as Algorithm 2.

Note that some of the resulting variables may be fractional. We round the fractional selection variable by assigning the node a to the cluster k with maximum x^a_k without violating the capacity constraints. Observe that the number of fractional elements will be smaller than the number of clusters, because each element will not be fragmented unless the cluster it is assigned to reaches its maximum capacity; after that instant, the cluster will take no additional data points. As a result, no other assignment will get fragmented on that cluster, and therefore, the number of fractional assignment will be less than that of clusters. Assuming that $N \gg K$, i.e., the number of data points is

Algorithm 2 Constrained-K-Means

Input:

$(z_a)_{a=1,...,N}$: data points formed by eigenvectors

K: number of clusters

$R = r_1, r_2, ..., r_N$: resource requirement of VNodes

$Cap = cap_1, cap_2, ..., cap_K$: capacity of each cluster

Output: Selection indicator x_k^a

Iteration $t \leftarrow 0$, randomly initialize $C_k^t \ \forall k$
Remaining resource requirement $Rm_{req} \leftarrow R$
Remaining capacity $Rm_{cap} \leftarrow Cap$
while $C^{t+1} \neq C^t$ **do**
    Clustering Assignment:
    Compute pairwise distance between data points and cluster centers $D = \{d_{11}, d_{12}, ..., d_{NK}\}$
    Ascending sort D to get $D_{asc} = \{d'_1, d'_2, ..., d'_{N*K}\}$
    for $j \leftarrow 1$ *to* $(N * K)$ **do**
        Choose point a and center point k associated with d'_j
        if $Rm_{req}^i < Rm_{cap}^h$ **then**
            $x_k^a \leftarrow Rm_{req}^i / Req_a$; $Rm_{cap}^h \leftarrow Rm_{cap}^h - R_{res}^i$; $Rm_{res}^i \leftarrow 0$
        else
            $x_k^a \leftarrow Rm_{cap}^h / Req_a$; $Rm_{req}^i \leftarrow Rm_{req}^i - Rm_{cap}^h$; $Rm_{cap}^h \leftarrow 0$
        end if
    end for
    Clustering Update:
    update the center points according to (2.13)
    $t \leftarrow t + 1$
end while
$P \leftarrow$ round X without violating capacity constraint.

Algorithm 3 Greedy Refinement Algorithm

Input:

K: Number of clusters
Initial assignment of VNodes to clusters
Rm_{cap}: Remaining capacity of each cluster
Cap: Maximum capacity for each cluster
Req: Resource requirement vector, remaining capacity

Output: Final assignment of VNodes to clusters

for $a \leftarrow$ selected virtual node **do**
    assume node $a \in$ cluster k
    $ED[a]_k|_{k=1,\ldots,K} \leftarrow 0$
    for $b \leftarrow 1$ *to* N **do**
        if node $b \in$ cluster k' **then**
            $ED[a]_{k'} = ED[a]_{k'} + t_{ab}$
        end if
    end for
    $k' = argmax\{ED[a]_k$ **s.t.** *capacity constraint*$\}$
    move a from cluster k to k'
    update $Rm_{Cap}^{k'}$
end for

significantly greater than the number of clusters, we expect that this greedy rounding scheme will have only relatively small negative impact.

2.3.3 Partitioning Refinement

In order to improve upon the partition obtained by the Constrained-k-means algorithm, we employ a greedy refinement (GR) method that employs simulated annealing (SA). The GR algorithm is proposed in [38], which improves the Kernighan-Lin (KL) algorithm [39] to refine the bisection of a graph by iteratively swapping the pair of vertices that would most significantly reduce the edge cut until a local minimum is reached. The GR algorithm extends the KL algorithm so as to handle vertex weights, refines the multi-way partitioning and improves the running time.

For completeness, we present the GR algorithm as Algorithm 3. Given an existing partition of the virtual nodes, the nodes are checked in a random order. Consider node a in cluster k. Denote $ED[a]_k$ as the total traffic between a and its neighbors that belong to cluster k (where we allow $k = k'$). The algorithm moves vertex a to the cluster with

Algorithm 4 New Point Generation

Input:
K: Number of clusters
n_{exc}: Number of nodes to exchange
Initial assignment of VNodes to clusters
Req: Resource requirement vector
Cap: Maximum capacity vector
Output: P: Final assignment of VNodes to clusters

```
for a ← random permutation from 1 to n_exc do
    node a ∈ cluster k
    if node a has neighbor ∈ cluster k' and
        Cap_k + Req_a < Cap'_k then
        move node a from cluster k to k'.
    end if
end for
for t ← 1 to t_ref do
    refine the partitioning via Greedy Refinement
end for
```

the highest value $ED[a]_k$ (or keeps it in the same cluster if it happens that $h = l$).

We now integrate this GR algorithm within a new point generation phase of SA: each time we randomly move a small number of nodes n_{exc} from one cluster k to another k', such that (1) the node that is moved from k to k' should be on the "brink" (i.e., it should have at least one neighbor in the new cluster k'), and (2) this movement does not violate the capacity constraints of cluster k'. The exchange aims to introduce perturbation to the current solution so as to escape local minima. The procedure to generate new points is detailed in Algorithm 4. After the exchange is completed, we execute several iterations of the Greedy Refinement algorithm to refine the result.

The GR algorithm constructs a solution that represents a local optimum. The total outgoing weight of this solution is considered as the energy function for the SA algorithm. The SA algorithm will decide whether to accept this point or not. Since the solution passed to SA is already a local optimum point obtained by the GR algorithm, the SA moves around the local optimum points to find the final solution. This operation is more efficient than the naive implementation of randomly generating new points and letting the SA algorithm decide which solution to take.

Running time: The overall algorithm (Algorithm 1) consists of three steps, namely,

computing the eigenvectors of the graph Laplacian, constrained k-means, and graph refinement. The computation of the eigenvectors can be completed in $O(N^3)$ time. For the constrained k-means, the clustering assignment subproblem can be solved in $O(KN \log N)$ time, where K is the number of clusters, and the cluster update problem can be solved in $O(kn)$ time. Let t_c be the number of iterations for constrained k-means to converge; then, the total running time for the constrained k-means is $O(Kt_cNlgN)$. Using an adjacency table, each iteration of the refinement phase can be completed in time $O(E)$. If t_r iterations are needed, the refinement phase takes time $O(t_rE)$. Overall, this algorithm runs in $O(N^3 + Kt_cN \log N + t_rE)$ time.

2.4 Experiments and Evaluation

We now present the results of experiments we have conducted to evaluate the performance of spectral clustering (SC)-based algorithms. We use two methods to refine the partitioning: solely based on the GR algorithm (referred to as SC) as well as the SA-based refinement approach (SC-SA) we described previously in this chapter. We compare the results to those obtained using a Gomory-Hu tree [47], the Metis [37], and the ILS method [41].

To evaluate the performance of our approach, we generate topology as follow: the virtual node is pairwise connected, and each node and link is assigned a weight to represent the resource and traffic requirements, respectively, and we also specify the maximum available capacity for each cluster. For each randomly generated topology, the vertex weight is uniformly distributed in $(1, 2)$, the edge weight is uniformly distributed in $(5, 20)$. Our goal is to partition the network into $K = 3, 4, 7$ clusters. For each cluster, we set the maximal capacity to 105% of the average weight of each cluster, i.e. the weight we place onto each cluster if we can have a perfect load balancing of the nodes. For the ILS algorithm, we set the number of iterations to be 5×10^4 and for each iteration, exchange 40% of nodes in each cluster. We also set $n_{exc} = 15$ for new point generation phase in SA, and we run 3 iterations of GR-algorithm to refine the partitioning result. We use two performance metrics: the inter-cluster traffic ratio (ITR), i.e., the ratio of the inter-cluster traffic to the total amount of traffic, and the running time of each algorithms. We run the simulation for 50 times, calculate the mean and confidence interval of our result and present it as follow:

Figures 2.3 and 2.4 plot the ITR and running time, respectively, against the number of virtual nodes and for $K = 3$ clusters. For the SA algorithm, we set the initial temperature to $Temp = 10^4$ and the maximum number of iterations to 600. We observe that spectral clustering with SC-SA method is strictly better than the other algorithms in terms of inter-cluster traffic minimization. Compared with Metis, it reduces the inter-cluster traffic by 6-9% percent, with an average improvement of 8.3%. Compared with Gomory-Hu tree (respectively, ILS), inter-cluster traffic is reduced by as much as 11% (respectively, 9.7%), with an average improvement of 10% (respectively, 8.2%). Also, compared with the SC only, SC-SA based refinement produces an improvement of 1.6% on average. In terms of running time, with a small number of virtual machines, the SC algorithm has the similar performance with Metis while the running time for SC-SA stays close to the Gomory-Hu tree method, about one magnitude larger than the two above, while ILS takes still one magnitude longer. When there are more virtual nodes, we see that the GH-Tree have a similar performance with the ILS, about an order of magnitude larger than the SC-SA, while SC takes less running time by an order of magnitude than SC-SA, and Metis takes yet another order of magnitude less than SC.

The second set of simulation experiments is to partition the virtual request into $K = 4$ clusters, and the results are shown in Figures 2.5 and 2.6. We kept the initial temperature for SA to $Temp = 10^4$ and the maximum number of iterations as 600. The SC algorithm produces clustering solutions that, in terms of inter-cluster traffic, outperform those produced by the Metis, Gomory-Hu tree, and ILS schemes by 4.5%, 6.5%, and 4.7%, respectively, on average. The SC-SA algorithm further reduces inter-cluster traffic by 1.1% on average, compared to SC. The running time results are similar to the experiments with $K = 3$ above.

Finally, Figures 2.7 and 2.8 plot the results of the third set of simulation experiments where we set $K = 7$. The initial temperature for SA was set to $Temp = 10^5$ and the maximum number of iterations to 600. The results are similar to those of the first two experiments, in that, on average, the SC algorithm performs 4.5% better than Metis, 6.1% better than Gomory-Hu tree, and 4.8% better than ILS. Also, compared with SC, the SC-SA algorithm reduces inter-cluster traffic by a further 0.7% on average. In terms of running time, the relative behavior of the five algorithms is also similar to the last two experiments.

From this set of simulations, we conclude that the spectral clustering method with

SA refinement produces the best solutions in terms of minimizing the inter-cluster traffic. It also compares favorably to existing clustering approaches based on ILS and Gomory-Hu tree, in terms of running time. When we compare with METIS, we found out its performance is at trade-off with Metis: the spectral clustering based algorithm achieve a better performance in minimizing the inter-cluster traffic, while Metis achieves a lower execution time.

To further evaluate the performance of our proposed algorithm, we compare the performance of our algorithm with METIS on three different kind of topologies with different partitioning settings. Among entire set of simulation, we selectively present three groups of evaluation results that would illustrate the performance.

First, we evaluate the result on random modular graph [34]. This intends to capture the pattern that the communication took place within one groups is more intensive than that of others. We assume the topology contains 20 clusters with equal size, and the probability of connecting one pair of node within the same cluster is 80%, while connecting one pair node from two different cluster is 20%. We generate the topology using this model, and the traffic on each link follows a Gaussian distribution, with a mean of 100 and variance of 25. We partition the generated graph into 3 clusters, and capacity constraint is set to 120% of the average. This setting for the capacity constraint tries to capture the condition that abundant computing resources in the underlying network is provided for each cluster such that a more flexible partitioning of the virtual request is allowed. The result is presented in the Figure 2.9.

In this set of simulation, we can see that inter-cluster traffic is minimized by SCSA with an additional 2.7% on average when compares with METIS. However, when we compare the result between METIS and SC, we can see that METIS achieves a better result on smaller request by as much as 9%, while the SC achieves a better result on a larger request by around 3%. On average, METIS can bring about 3% of improvement than SC for traffic minimization.

Next, we generate the topology following the Waxman model [55]. The Waxman model is a popular kind of model for the modelling of the Internet, especially for the intra-domain network [48]. In a Waxman model, nodes are randomly placed on a rectangle area, and the connectivity probability between a pair of nodes are based on their Euclidean distance, more formally, $p = \alpha \exp(-d/(\beta * L))$, where d is the Euclidean distance between one pair of nodes, and the L is maximum distance allowed. The α and

β are two parameters that will define the connectivity pattern. More specifically, α will define the overall connectivity probability, a higher α will result in a denser connectivity, while β restricts the probability of connection based on the distance, i.e. with a larger β tends to increase the chance of connection for a pair of nodes that are far from each other, while a lower β tends to prevent such connection. In this experiment, we use $L = 100$, $\alpha = 0.05$, and $\beta = 0.3$ to define the connectivity of the virtual request. For the partitioning setting, we divide the virtual request into 4 clusters with different capacities, each cluster with 20%, 20%, 30%, 30% of the total weight of the request. Maximum violation for the capacity is defined to be 5%. This setting may comply with the situation where the distribution of the computational resource is imbalanced, such as heterogeneous resources, or imbalanced workload on different domains. The simulation setting is shown in Figure 2.11.

As we can see from the simulation results, the SCSA achieves an improvement of 5% compared with METIS for ITR minimization, while SC delivers a 2.5% improvement compared with METIS.

Finally, we evaluate this algorithm on the virtual request whose degree distribution exhibits power law, which characterizes the degree distribution of router-level and AS-level Internet graphs [23]. We use Barabási-Albert (BA) model [6] to obtain a topology that follows the power-law. It starts with a small connected component, and then probabilistically attach new vertex to the existing vertex following preferential attachment, i.e. the likelihood of connection depends on the degree of the existing vertex. We start with 5 connected vertex and ended up with a topology consisting of 50 to 400 vertices. We partition the graph into four clusters with equal capacity, while allows a cluster to exceed its capacity limit by 5%, the simulation result is shown in Fig.2.13.

Compared with METIS, the simulation result suggests an average improvement of 34% in term of ITR minimization for SCSA, with a maximum improvement of 43%, while SC brings forth an improvement of 14% compared with METIS on average.

The running time on three different type of topology is similar to the previous set of simulation. As it is shown in Fig. 2.10, Fig. 2.12 and Fig. 2.14. METIS requires less execution time compared with SCSA, by two orders in general. Compared with SC, we see that their performance is similar on a request with small scale, while on a larger request, METIS will still work better.

From our whole set of simulation, we can conclude that this proposed algorithm can

achieve a better performance in term of traffic minimization than METIS, while METIS can achieve a better running time. Notably, for a request consists of 400 nodes, the partitioning can be done by our algorithm in about 1 seconds. This would imply that the setup delay caused by our algorithm would be acceptable in most cases.

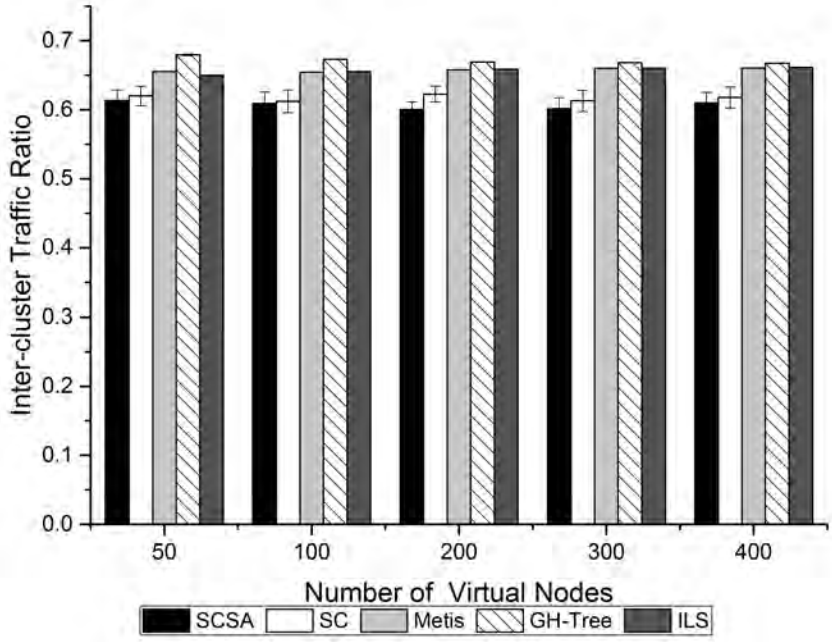

Figure 2.3: Inter-cluster Traffic Ratio for K=3

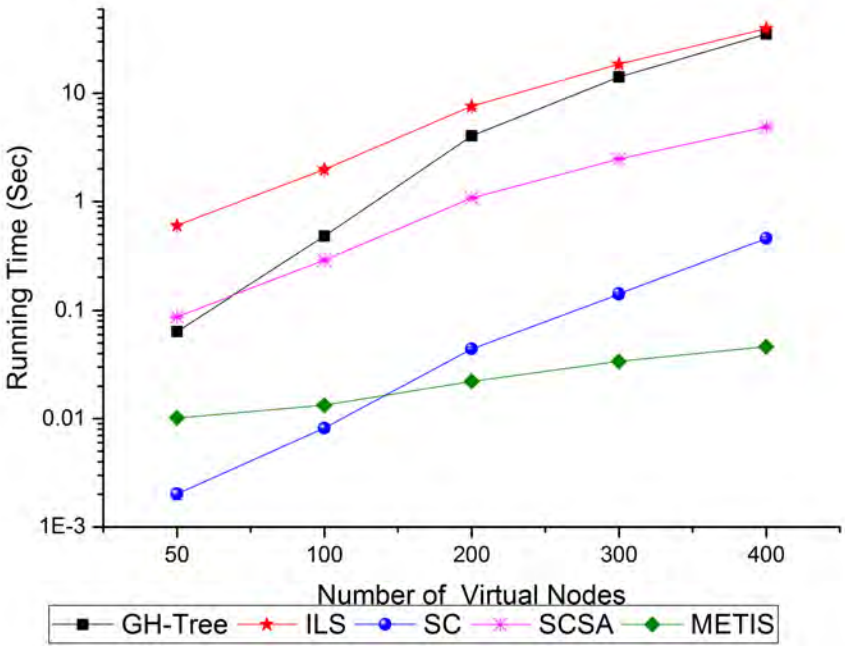

Figure 2.4: Running Time for K=3

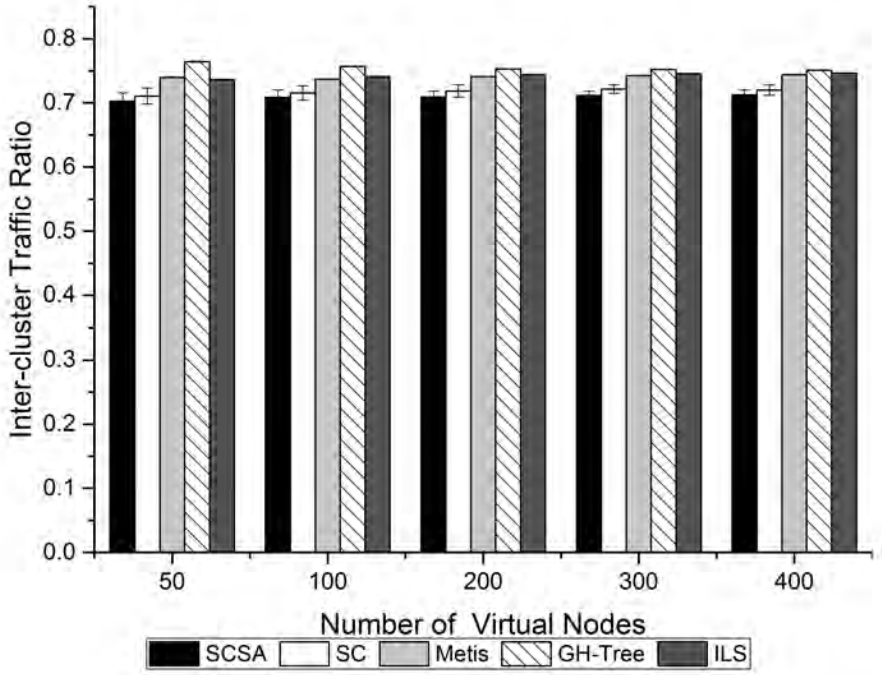

Figure 2.5: Inter-cluster Traffic Ratio for K=4

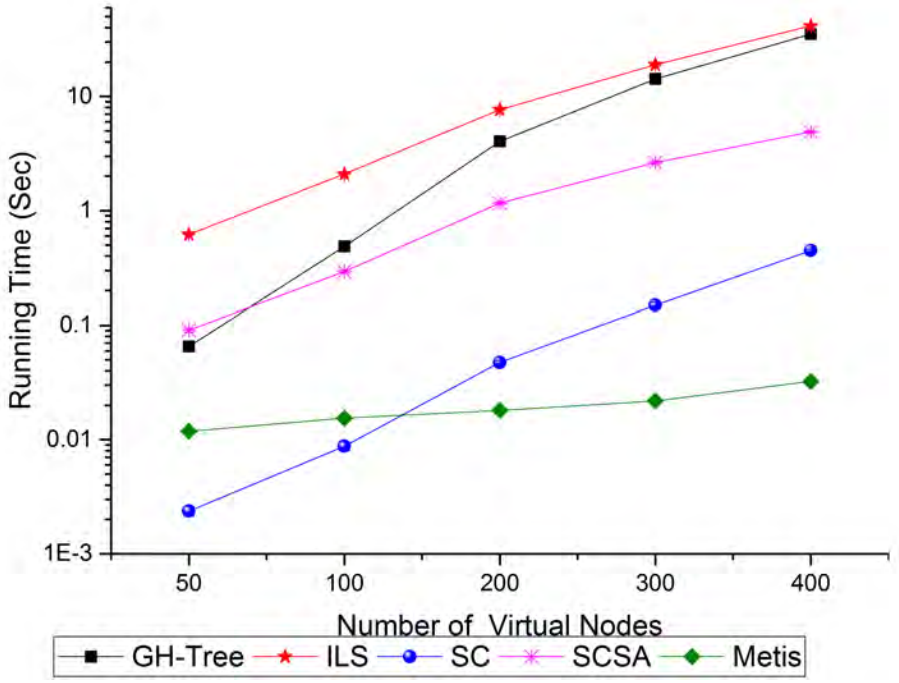

Figure 2.6: Running Time for K=4

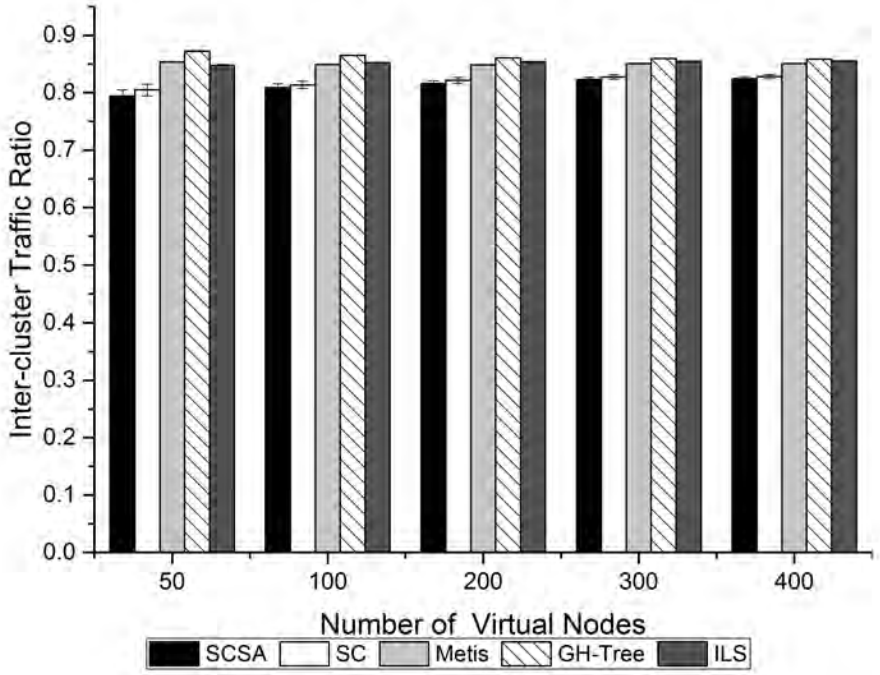

Figure 2.7: Inter-cluster Traffic Ratio for K=7

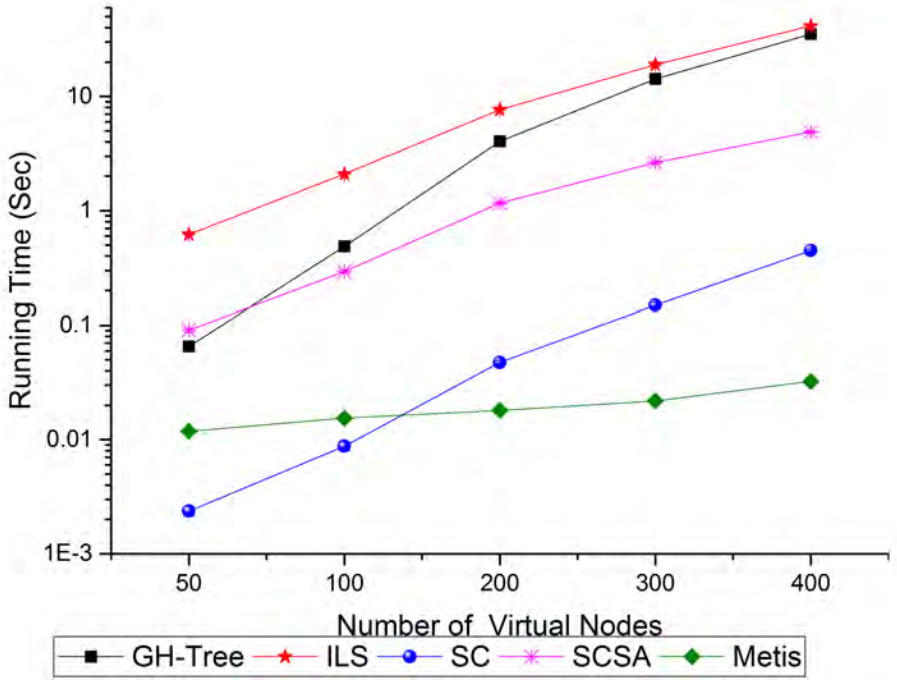

Figure 2.8: Running Time for K=7.

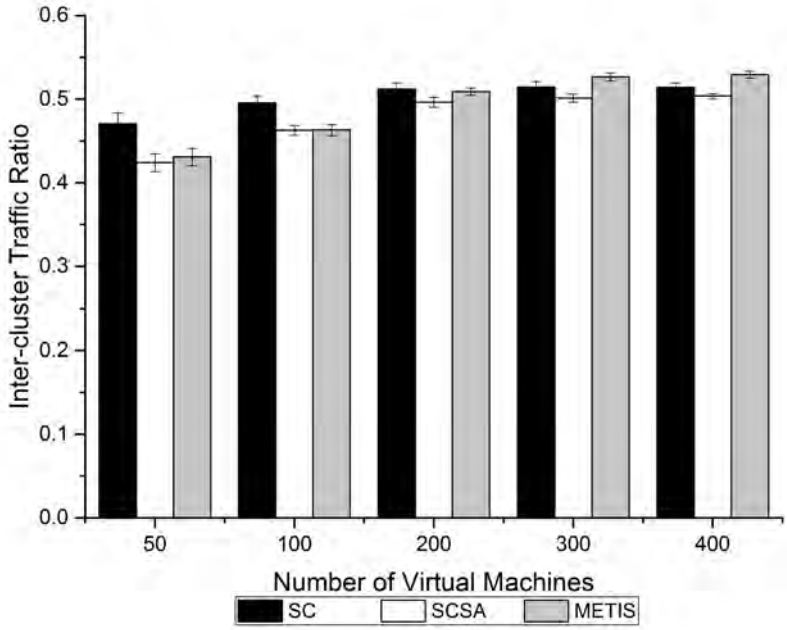

Figure 2.9: Inter-cluster Traffic Ratio for Modular Pattern

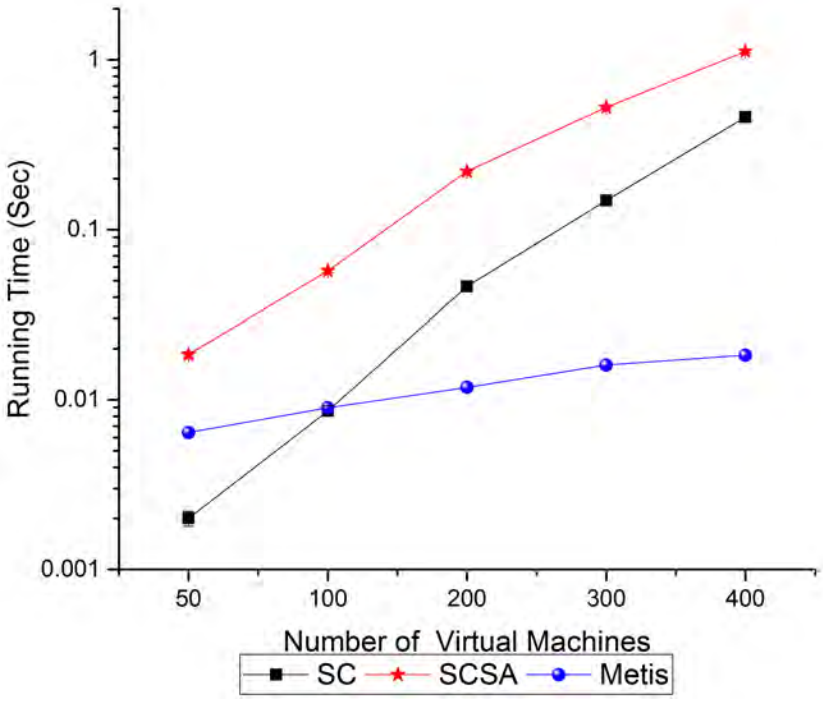

Figure 2.10: Running Time for Modular Pattern

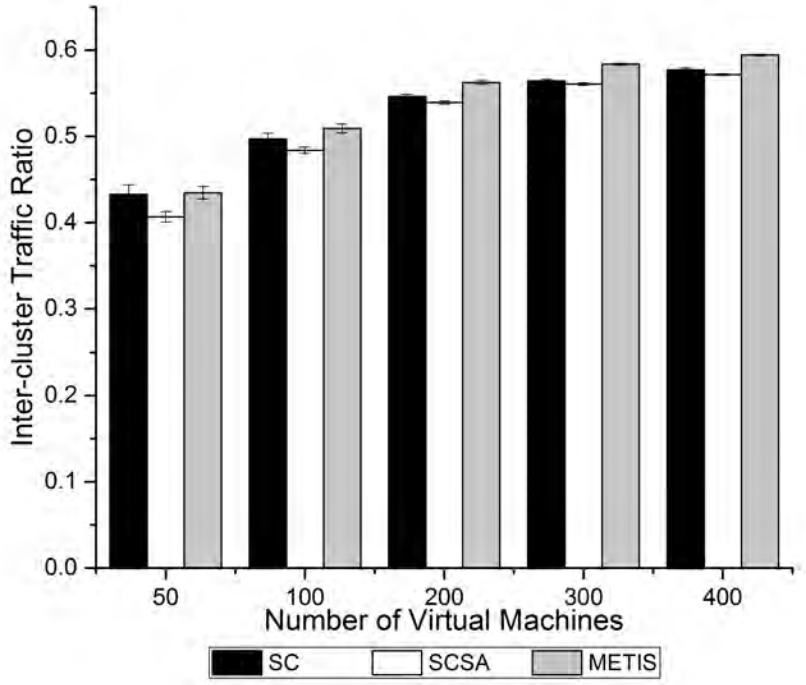

Figure 2.11: Inter-cluster Traffic Ratio for Waxman Model Pattern

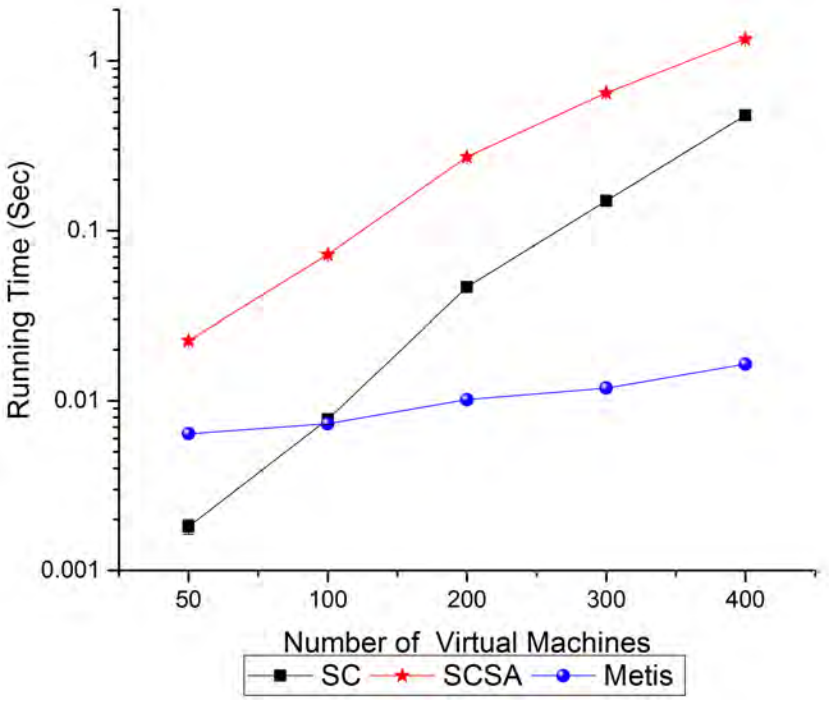

Figure 2.12: Running Time for Waxman Model Pattern

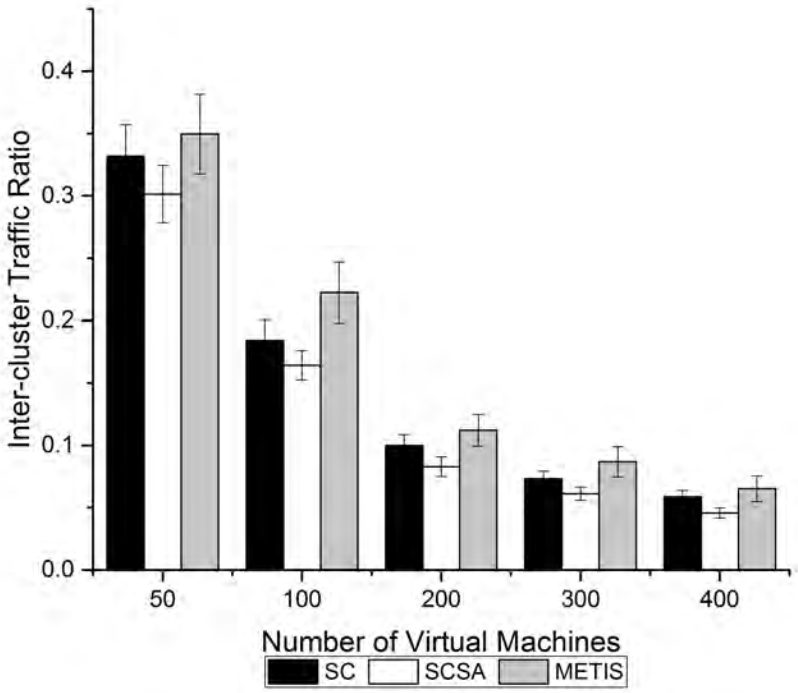

Figure 2.13: Inter-cluster Traffic Ratio for BA-Model Pattern

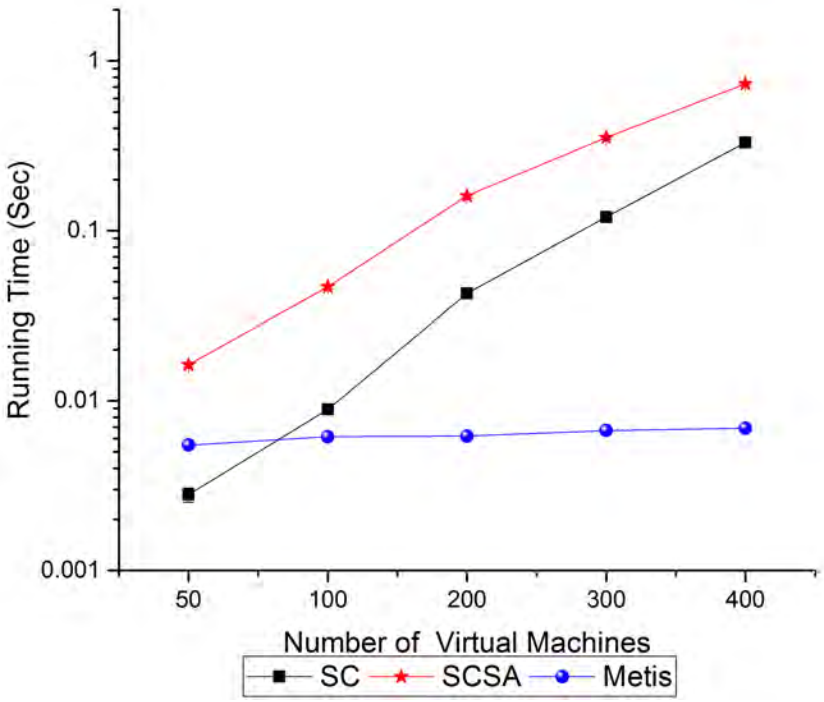

Figure 2.14: Running Time for BA-Model Pattern.

Chapter 3

Virtual Network Reconfiguration with Migration Cost Consideration

In this chapter, we present the design of an algorithm that will handle the online nature of the virtual network requests by reconfiguring the virtual network requests. Such a reconfiguration may help to improve network performance by remapping a subset of virtual nodes or links to better align the allocation of resources to current network conditions. In Chapter 3.2, we formally define the Virtual Network Reconfiguration (VNR) problem and introduce the augmented graph model we use in developing the formulation. We present the algorithm we have developed for the VNR problem in Chapter 3.3, and we evaluate its performance in Chapter 3.4.

3.1 Related Work in Virtual Network Reconfiguration Problem

A method to reconfigure the VN requests to maximize the InP revenue has been proposed in [61]. In this study, emphasis is on the embedding of virtual links under the assumption that the traffic of a virtual link may be split and carried over several substrate paths. Consequently, the authors propose a reconfiguration scheme that dynamically adjusts the splitting ratio of a virtual link or sets up a new path in the substrate network. Although virtual link splitting and migration improves the acceptance ratio for VN requests, migration of virtual nodes is not considered in this work.

A proactive reconfiguration algorithm, referred to as global marking algorithm, was proposed in [62]. In the reconfiguration phase, the workload of the substrate nodes and links is examined. If a given substrate node or link is overloaded, then all virtual nodes and virtual links mapped on top of that particular substrate node or link, respectively, are marked for remapping. During the remapping phase, marked virtual nodes and links are re-assigned to the SN. This approach may incur unnecessary reconfigurations: often, only a fraction of the virtual nodes on a stressed substrate node need to be migrated, whereas this method tends to reconfigure all of them. Also, for congested substrate links, solely reconfiguring the mapping of the virtual links without migrating the virtual nodes causing the overload may not fully alleviate the congestion.

Another reconfiguration scheme to maximize InP revenue was proposed in [52]. This approach takes into account the migration overhead so as to limit the service disruption caused by reconfiguring the virtual nodes. Reconfiguration is triggered whenever a VN request is blocked, and follows the solution of a MIP problem. Since this method relies on solving the MIP problem, it may not scale to large problem sizes or may not be appropriate whenever a low reconfiguration delay is required.

Based on the observation in [22] that VN request rejection is mainly caused by bandwidth shortage, a reactive VN reconfiguration algorithm was presented in [19]. The algorithm aims at improving the VN request acceptance rate while limiting the number of virtual nodes to be migrated, so as to minimize the overall reconfiguration cost. Virtual nodes are selected for migration based on the number of congested links along which they route their traffic. Selected virtual nodes are iteratively reconfigured until either the incoming VN request is successfully embedded or the number of virtual nodes reconfigured exceeds the given threshold. This work solely considers congested links and does not account for overloaded substrate nodes. Also, it does not take into consideration the magnitude of traffic demands originating/terminating at the virtual nodes. This may result in a situation whereby virtual nodes with little traffic (and hence, small impact on the SN) are selected for migration just because they happen to use congested paths. As an extension to the work in [22], a VN reconfiguration strategy was developed in [19] to handle the scale-up/down of VN requirements requested by users. This study used a genetic algorithm to minimize the combined cost of embedding and reconfiguration.

A scheme to reconfigure the VN within an evolving substrate network was proposed in [14]. In this work, reconfiguration is triggered in response to changes in the underlying

SN, i.e., whenever substrate nodes and/or links are added or removed. The objective is to reconfigure the VN such that delay constraints are preserved while migration overhead is minimized. To this end, a heuristic algorithm was proposed to relocate virtual nodes that are affected by the changes or violate the delay constraints.

3.2 Problem Definition

We model the SN as a weighted, undirected graph $G^s = (V^s, E^s)$. The vertices of G^s stand for the substrate nodes, while the edges of G^s represent the substrate links. Each node A and edge (A,B) on G^s are weighted, and we let Cap_A denote the resource (e.g., CPU) capacity of the substrate node A, and Cap_{AB} denote the bandwidth capacity of substrate link (A,B).

Likewise, we model a VN as a weighted, undirected graph $G^v = (V^v, E^v)$, whose vertices and edges represent virtual nodes and links, respectively. The weight Req_a of vertex a reflects the resource (e.g., CPU) requirement of the virtual node, while the weight t_{ab} denotes the traffic demand of the virtual link (a, b)

In addition, each virtual node may specify additional constraints, including the type of the substrate nodes they may be mapped onto, geographical constraints, etc., such that a virtual node may be placed only on a specific group of substrate nodes. We denote the set of substrate nodes that may support a virtual node a as $\Theta(a) \subseteq V^s$.

We assume that there exists a mapping of virtual nodes and links to the substrate network, $\mathcal{F} : G^v \rightarrow G^s$. We also assume that the resource requirements Req_a and traffic demands t_{ab} of the VNs evolve over time such that the current mapping $\mathcal{F}$ is not representative of the current state of the network. Although our work is agnostic with respect to how reconfiguration is triggered (e.g., whether it is performed periodically or is initiated as soon as a performance measure crosses a predefined threshold), our focus is on updating the mapping $\mathcal{F}$ to align it with current network conditions.

3.2.1 Reconfiguration Objectives

Our goal is to develop an online VN reconfiguration algorithm that remaps part of the VN requests so as to balance the load on the substrate nodes and links. Since the migration is more expensive than the reconfiguration of virtual links [62], we also aim to limit the number of virtual nodes that have to be migrated, so as to keep the reconfiguration cost

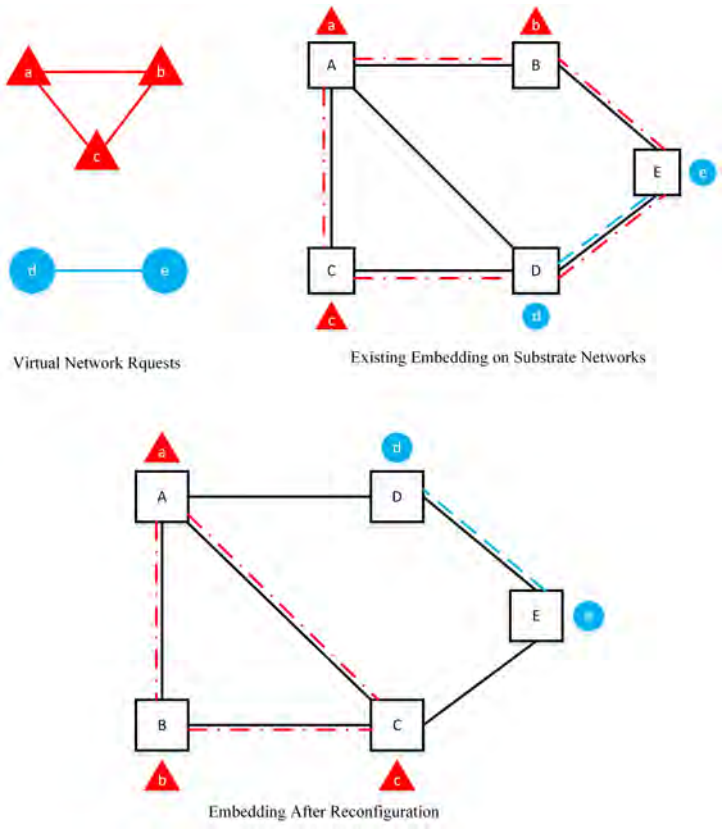

Figure 3.1: Reconfiguration of virtual network requests

low and minimize service disruption. A reconfiguration example is shown in Figure 3.1. The top part of the figure shows the two VN requests and the original embedding of the requests onto the substrate network, whereas the bottom part of the figure shows the new embedding of the two requests after reconfiguration has taken place.

We view this VNR problem as a multi-objective optimization problem, with three metrics to be minimized: substrate link utilization, substrate node workload, and the number of virtual nodes to be migrated. Specifically, we take link utilization as the primary objective and bound the other two, such that we formulate the goal in this form:

> *Minimize the maximum link utilization λ, under two constraints: resource utilization of each substrate node does not exceed $\rho\lambda$, and the total number of virtual nodes being migrated does not exceed a threshold M.*

M and ρ are parameters defined by the network operator. Parameter ρ may be used to adjust the tradeoff between minimizing the utilization of substrate links and nodes.

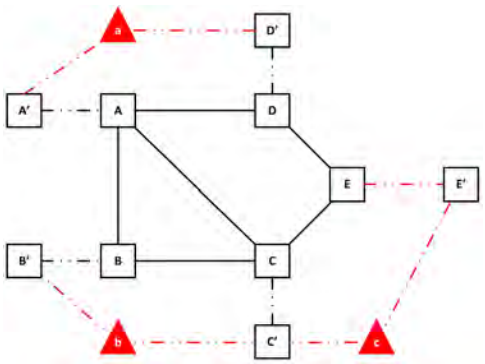

Figure 3.2: Augmented graph

3.2.2 Augmented Graph

Our problem formulation makes use of the augmented graph presented in [18]. We start with the graph G^s that represents the substrate network and follow these steps to construct the augmented graph G^{aug} (refer also to Figure 3.2). For each substrate node A, we create a mirror node A', as well as an edge (A, A') with weight equal to the capacity of node A, Cap_A. For each virtual node a, we also create a corresponding node in the augmented graph. Recall that $\Theta(a)$ is the subset of substrate nodes to which virtual node a may be mapped. Therefore, for each substrate $A \in \Theta(a)$, we create an edge (a, A') between the virtual node a and the mirror node of A. For instance, in Figure 3.2 we assume that $\Theta(a) = \{A, D\}$, hence the augmented graph contains the edges (a, A') and (a, D'). The capacity of these edges is set to infinity.

In addition to the weight (capacity), we associate a cost with each edge in the augmented graph, as follows: the cost on all edges except the ones between virtual nodes and mirror substrate nodes is zero. If, in the current configuration, a virtual node a is mapped on, say, substrate node A, then the cost of edge (a, A') is also zero. Otherwise, the cost of the edge $(a, D'), A \neq D \in \Theta(a)$, is the reciprocal of the total traffic of virtual node a, i.e., $1/\sum_b t_{ab}$.

3.2.3 MIP Formulation

Using the augmented graph defined above, we formulate the VNR problem as the following mixed-integer programming (MIP) problem.

Decision Variables:

x_A^a: binary variable, indicating whether virtual node a is mapped onto substrate node A.

f_{AB}^{ab} : flow variable, indicating the fraction of traffic between virtual nodes a and b that is mapped onto substrate link (A, B).

MIP Formulation:

$$\min \ \lambda \tag{3.1}$$

$s.t.$

$$\sum_{A \in \Theta(a)} f_{aA'}^{ab} - \sum_{A \in \Theta(a)} f_{A'a}^{ab} = 1, \quad \forall a, b \in V^v, (a, b) \in E^v \tag{3.2}$$

$$\sum_{B \in \Theta(b)} f_{bB'}^{ab} - \sum_{B \in \Theta(b)} f_{B'b}^{ab} = -1, \quad \forall a, b \in V^v, (a, b) \in E^v \tag{3.3}$$

$$\sum_{A \in V^S} f_{AB}^{ab} - \sum_{A \in V^S} f_{BA}^{ab} = 0, \quad \forall a, b \in V^v, (a, b) \in E^v, A, B \in V^s \tag{3.4}$$

$$\sum_{(a,b) \in E^v} t_{ab} f_{AB}^{ab} \leq \lambda Cap_{AB}, \quad \forall A, B \in V^s \tag{3.5}$$

$$\sum_{a \in V^v} Req_a x_A^a \leq \rho \lambda Cap_A, \quad \forall A \in V^s \tag{3.6}$$

$$\sum_{b \in V^v} f_{aA'}^{ab} = \sum_{b \in V^v} x_A^a, \quad \forall a, b \in V^v, \ \forall A \in V^s \tag{3.7}$$

$$\sum_{a \in V^v} (1 - x_A^a) \leq M, \ \forall a \in V^v, A = Ext(a) \tag{3.8}$$

$$\sum_{A \in \Theta(a)} x_A^a = 1, \ \forall a \in V^v \tag{3.9}$$

$$0 \leq f_{AB}^{ab} \leq 1 \tag{3.10}$$

$$x_A^a \in \{0, 1\} \tag{3.11}$$

Remarks:

The objective of the MIP is to minimize the maximum utilization of any link in the SN.

Constraints (3.2)-(3.4) are the flow related constraints. Assume there exists a virtual link (a, b) between virtual nodes a and b, i.e., that the traffic between them $t_{ab} > 0$. Constraint (2.2) specifies that all traffic between a and b must pass pass through the mirror nodes that are connected to virtual node a in the augmenting graph. Similarly, constraint (2.3) ensures that all traffic will be routed to virtual node b via mirror nodes connected to that node in the augmented graph. Constraint (2.4) is the flow conservation constraint, and ensures that the net traffic in and out of a substrate node is zero.

Constraints (3.5) and (3.6) are the substrate link and node capacity constraints, respectively. Constraint (3.5) states that the total amount of traffic on a substrate link may not exceed λ times its capacity, while constraint (3.6) asserts that the resource requirement on a substrate node may not exceed $\rho\lambda$ times its capacity.

Constraint (3.7) maintains consistency between decision variables x and f, as it guarantees that, if virtual node a is mapped onto substrate node A, then all traffic associated with a will go through the augmented link between node a and the mirror node A'.

Constraint (3.8) ensures that the number of virtual nodes to be migrated does not exceed the threshold M. Here, $Ext(a)$ stands for the substrate node upon which virtual node a is mapped prior to reconfiguration. If, after reconfiguration, virtual node a is still placed upon substrate node $A = Ext(a)$, then the term $(1 - x_A^a)$ will be 0. Thus, the sum on the left hand side of the constraint equals the number of virtual nodes that are remapped due to reconfiguration.

Constraint (3.9) ensures that each virtual node is mapped to exactly one substrate node, while the last two constraints define the range of the decision variables.

3.3 Reconfiguration Algorithm

Since VNR is an NP-hard problem, to tackle it efficiently we decompose it into two subproblems, namely, virtual node selection and virtual node remapping, described below.

Virtual Node Selection. Inspired by the work of [18], we use an LP relaxation based approach to select the virtual node to be migrated. First, we relax the integral constraints on variables x_A^a, i.e., constraints (3.11), and solve the resulting linear programming problem.

Hence, virtual node selection is carried out by jointly considering the load on substrate links and nodes. Although LP problems are solvable in polynomial time, the computation time may be prohibitive for large-scale networks. To overcome this difficulty, we propose an approximation algorithm to obtain a near-optimal solution with small computation overhead. As a result, the delay to reach a reconfiguration decision may be kept low.

Once the LP problem is solved (optimally or using the approximation algorithm), we ascending sort variables x_A^a, where $A = Ext(a)$. Note that we can think of x_A^a as the likelihood that virtual node a is to remain in the same substrate node A. Consequently, we mark the M virtual nodes with the smallest value of x_A^a as the nodes to be migrated.
Virtual Node Remapping: We have developed an algorithm based on a random walk on a Markov chain to filter substrate nodes for the virtual nodes selected in the solution to the previous subproblem, and use a MIP to map them back to the substrate network. This remapping phase takes into account load balancing across the substrate nodes and links.

3.3.1 Virtual Node Selection

As we mentioned above, solving a large-scale LP problem is computationally expensive, and may result in long reconfiguration delays that are not acceptable in an online scenario such as the one we are considering. Furthermore, we note that solving the LP problem exactly is *not* necessary: our focus is not on the exact values of variables x_A^a but rather on their relative values, since the goal is to rank virtual nodes based on their likelihood to be migrated. With this in mind, we build upon the work of [36] to design an approximation algorithm to obtain a near-optimal solution to the relaxed LP.

Path-based Formulation

In order to develop the approximation algorithm, we first present a path-based formulation that is equivalent to the link-arc MIP formulation (3.1)-(3.11), and relax the integral constraints. The path-based formulation is based on the observation that, by construction, the augmented graph imposes a connection between the mapping of a virtual node and routing. Specifically, if there is traffic flow between a virtual node a and a mirror node A', this implies a mapping of a onto substrate node A.

For simplicity, we denote the edges of the augmented graph between a substrate node A and its mirror node A' as e^A, and its capacity as $c(e^A)$ (recall that this is equal to the

capacity of the corresponding substrate node). Similarly, we denote edges representing substrate links as e, and their capacity as $c(e)$. Let P_{ab} be the set of paths between virtual nodes a and b, and T_a denote the traffic originating at virtual node a, i.e., $T_a = \sum_{b \in V^v} t_{ab}$. We define the decision variables $x(p)$ as the amount of flow routed along path p. We also define r_p^a as a normalization factor which builds an association between the routing of the flow and the mapping of virtual nodes to substrate nodes: if routing flow along path p indicates that virtual node a is mapped onto substrate node A, then $r_p^a = Req_a/T_a$. Note also that, since the flow on a path p indicates the substrate node to which the virtual nodes generating that flow are mapped, it also indicates whether or not these virtual nodes are to be migrated. Therefore, we introduce binary parameter η_p^a defined as follows: if path $x(p)$ indicates that virtual node a is to be migrated, then $\eta_p^a = 1$, otherwise $\eta_p^a = 0$.

Based on these observations, we have the following path-based LP formulation of the relaxed version of the above MIP problem.

LP Formulation:

$$\min \ \lambda \tag{3.12}$$

$s.t.$

$$\sum_{p:e \in p} x(p) \leq \lambda c(e), \quad \forall e \in E^s \tag{3.13}$$

$$\sum_{p:e^A \in p} r_p^a x(p) \leq \rho \lambda c(e^A), \forall A \in V^s \tag{3.14}$$

$$\sum_{p \in P_{ab}} x(p) \geq t_{ab}, \quad \forall a, b \in V^v \tag{3.15}$$

$$\sum_{p,a} \frac{\eta_p^a}{T_a} x(p) \leq M \tag{3.16}$$

$$x(p) \geq 0, \quad \forall p. \tag{3.17}$$

Similar to the link-arc form, the objective function of the LP is to minimize the maximum utilization of the substrate links and nodes. Constraint (3.13) enforces the capacity constraints for substrate links, and is equivalent to constraint (3.5). Constraint (3.14) represents the capacity constraint for substrate nodes. Based on the information from

the augmenting graph, routing an amount $x(p)$ of traffic along this path determines the virtual nodes to be mapped to the substrate nodes at both ends of the path. Based on the definition of r_p^a, and combined with equation (3.18), one can verify that $\sum_{p:e^A \in p} r_p^a x(p)$ is the total amount of node resource requests placed on substrate node A from path p. By summing up over all paths p that contain this node A, constraint (3.14) is equivalent to constraint (3.6).

Constraint (3.15) ensures the traffic demand is satisfied. As this constraint considers only the paths p between two specific virtual nodes in the augmented graph, and only non-negative flow is routed along each path p, we conclude that constraints (3.2) to (3.4) are satisfied.

Constraint (3.16) is equivalent to constraint (3.8), which ensures that the total number of virtual nodes to be migrated does not exceed the given threshold M. Since $\eta_p^a/T_a x(p) > 0$ only when path p indicates migration of virtual node a, the sum of all paths associated with virtual node a indicates the fraction of virtual node a to be migrated; consequently, the sum over all paths and all virtual nodes, $\sum_{p,a} \eta_p^a/T_a x(p)$, indicates the total (non-integer) number of virtual nodes to be migrated.

In addition, based on equation (3.18) below, one can verify that with an integer solution, each virtual node is mapped onto one and only one substrate node, satisfying constraint (3.9).

Finally, to prove that this path-based LP formulation is equivalent to the MIP problem we discussed in Chapter 3.2 when we relax the integral constraint, we now show how to obtain the values of the decision variables of the MIP problem from the LP solution $x(p)$.

First note that x_A^a is no longer a binary variable but rather takes values in $[0, 1]$ and represents the fraction of traffic from virtual node a that goes through substrate node A, i.e., it is routed along the edge (a, A') in the augmented graph. Therefore, we may obtain x_A^a as the ratio of the traffic on that path to the total traffic originating at a:

$$x_A^a = \sum_{p:(a,A') \in p} x(p)/T_a \tag{3.18}$$

Similarly, f_{AB}^{ab} may be obtained by taking the ratio of the amount of traffic that passes through an edge (A, B) over the total traffic from virtual node a to b, i.e.,

$$f_{AB}^{ab} = \sum_{p:(AB) \in p, p \in P_{ab}} x(p)/t_{ab} \tag{3.19}$$

Equivalent Form–Minimum Cost Multi-Commodity Flow Problem

From our previous discussion, we know that the routing of traffic implies the mapping of virtual nodes. Therefore, we may then tackle the above LP as a minimum cost multicommodity flow (MCMCF) problem: we consider traffic between each pair (a, b) of virtual nodes as a commodity that needs to be shipped across the augmented graph, and our goal is to ship the maximum amount of commodity without violating cost and capacity constraints. To this end, we re-organize the previous path-based LP as follows.

MCMCF Formulation:

$$\max \ \lambda \tag{3.20}$$

$s.t.$

$$\sum_{p:e\in p} x(p) \leq c(e), \quad \forall e \in E^s \tag{3.21}$$

$$\sum_{p:e^A\in p} r_p^a x(p) \leq \rho c(A), \quad a : (aA' or A'a) \in p, \forall A \in V^s \tag{3.22}$$

$$\sum_{p\in P_{ab}} x(p) \geq \lambda t_{ab}, \quad \forall a, b \in N^v \tag{3.23}$$

$$\sum_{p,a} \frac{\eta_p^a}{T_a} x(p) \leq \sigma M \tag{3.24}$$

$$x(p) \geq 0, \quad \forall p \tag{3.25}$$

Constraints (3.21) and (3.22) are the capacity constraints, while constraint (3.24) is the cost constraint. To see this, recall the way we defined the edge cost for the augmented graph in Chapter 3.2: non-zero cost edges only exist between each virtual node and mirror substrate nodes to which this virtual node is not currently mapped. Therefore, routing an amount $x(p)$ of flow along path p that contains a new substrate node (and hence, implies migration of virtual node a) implies a cost of $x(p)/T_a$; otherwise, the cost is zero. As a result, the cost of routing all the flow is given by the left-hand side of (3.24). Thus, constraint (3.24) ensures that the total cost of sending commodities across the network is kept below a predefined threshold σM, where σ is a parameter (more on σ shortly).

Note that the MCMCF formulation aims to maximize the demand (scaled by λ in (3.23)) to be routed across the augmented graph with a fixed amount of capac-

ity, whereas the LP formulation aims to maximize the capacity (scaled by λ in (3.13) and (3.14)) to carry the given demand. Therefore, an optimal solution $(\lambda^\star, x^\star(p))$ to one problem will also be an optimal solution to the other, as long as we let $\sigma = \lambda^\star$ in (3.24). Since, however, we do not know $\lambda^\star$ *a priori*, we will discuss later in this chapter how to obtain a value for σ that is close to $\lambda^\star$.

In [36], a fully polynomial time approximation scheme (FPTAS) has been proposed to solve an LP-based MCMCF approximately. Specifically, the FPTAS obtains a solution that is within a factor of $(1 + \omega)$ of the optimal solution and runs in time $\widetilde{O}(\omega^{-2}|E|^2)$, where E is the number of edges in the augmenting graph.

Our MCMCF formulation above is similar to the MCMCF problem in [36], except that we have two sets of capacity constraints instead of one. The existence of two sets of capacity constraints means that the FPTAS of [36] may not be directly applied to solve our problem. In the remainder of this chapter, we show how to extend the FPTAS of [36] to obtain one for our problem.

Dual Problem

We start by stating the dual form of the MCMCF problem.

Decision Variables:

$l(e)$: Associated with constraint (3.21), interpreted as the length of substrate edge on augmented graph

$l(e^A)$: Associated with constraint (3.22), interpreted as length of the edge between a substrate and mirror substrate node

$z(ab)$: Associated with constraint (3.23), interpreted as the shortest path between a and b

ϕ: Associated with constraint (3.24), interpreted as the penalty factor for migration.

Dual Problem Formulation

$$\min_{l,\phi} D(l,\phi) = \sum_{e} c(e)l(e) + \rho \sum_{e^A} c(e^A)l(e^A) + \sigma M \phi \tag{3.26}$$

$s.t.$

$$\sum_{e \in p} l(e) + r_p^a l(e^A) + r_p^b l(e^B) + (\frac{\eta_p^a}{T_a} + \frac{\eta_p^b}{T_b})\phi \tag{3.27}$$
$$\geq \ z(ab), \ \forall p \in G^{aug}, \ \forall a, b$$

$$\sum_{ab} t_{ab} z(ab) \geq 1 \tag{3.28}$$

We denote the dual objective as $D(l,\phi)$ in (3.26), which consists of two parts: total length of the augmenting graph $\sum c(e)l(e) + \rho \sum c(e^A)l(e^A)$ weighted by the edge capacity, and weighted migration penalty $\sigma M \phi$. Therefore, the dual problem can then be seen as an assignment of length functions $l(e)$, $l(e^A)$ to the edges of the augmented graph, and penalty factor ϕ such that the weighted sum $D(l,\phi)$ is minimized.

We define the length of a path between virtual node a and b under the length function as follows:

$$\sum_{e \in p} l(e) + r_p^a l(e^A) + r_p^b l(e^B) + (\frac{\eta_p^a}{T_a} + \frac{\eta_p^b}{T_b})\phi \tag{3.29}$$

More specifically, if path p implies no migration for either a or b, then the distance

between a and b will be measured by the length function; otherwise, for each virtual node to be migrated, an additional penalty of ϕ/T is incurred. When we have a feasible solution to the dual problem, we interpret $z(ab)$ as the shortest path between virtual nodes a and b under the length function and penalty factor.

We now design a primal-dual approximation algorithm to solve the MCMCF problem.

FPTAS Algorithm to Solve the MCMCF

The FPTAS for the MCMCF problem is shown as Algorithm 1, and is based on the FPTAS in [36]. At a high level, the algorithm operates as follows. We assign an initial length to all edges, and then we iteratively route flow along the shortest path. We determine δ and precision factor ϵ in the same way as in [36].

The length of a path p is defined in (3.29). Each time we send flow along path p, the length of each edge along this path increases, thus reducing the likelihood that subsequent flow will follow this edge. This helps to spread traffic evenly among all edges and increases the overall throughput.

The algorithm proceeds in phases, each phase having $|V^v|$ iterations. During the a^{th} iteration, we route the flow along the shortest path between virtual node a and all its neighbors in the VN under the current length function and penalty factor. For example, if virtual nodes a and b are neighbors in the VN with traffic demand t_{ab}, then in the iteration corresponding to node a, we route t_{ab} amount of flow between virtual nodes a and b.

Without loss of generality, we make the following two assumptions:

- In each phase, the amount of traffic generated by each virtual node is non-decreasing. In other words, once at a certain iteration we route all traffic between virtual node a and all its neighbors, no virtual node in a subsequent iteration will generate more traffic than a.

- The total number of virtual nodes we allow to be migrated is at least two.

The first assumption can be achieved easily by considering virtual nodes in increasing order of the traffic they generate. As for the second assumption, if we allow no more than one virtual node to be migrated, then we may simply solve this problem by a greedy algorithm that examines the performance of migrating one virtual node to another substrate node, and pick the node that produces the best result.

Algorithm 5 FPTAS for MCMCF

Input:

G^{aug}: augmented graph, $c(e)$: substrate link capacity

$c(e^A)$: substrate node capacity

t_{ab}: traffic between a and b

r_p^a: normalizing factor, T_a: total traffic from a

ϵ: precision factor, ρ: balancing factor

σ: parameter for initial length

Output: λ: maximum utilization of substrate link

$x(p)$: assignment of flow to the substrate network

Initialize $l(e) = \delta/c(e),\ l(A) = \delta/\rho c(e^A), \phi = \delta/M$;
while $D(l, \phi) < 1$ **do** ▷ Phase
  for $a = 1, 2, \cdots, |V^v|$ **do** ▷ Iteration
    $t_{ab}^r = t_{ab}, b = 1, ..., |V^v|$ ▷ Remain Traffic
    $c^r(e) := c(e)$ ▷ Remain Capacity
    while $D(l, \phi) < 1$ *and* $t'_{ab} > 0, \exists\ k$ **do** ▷ Step
      $SPT_a :=$ shortest path tree rooted on node a
      for all b with $t_{ab}^r > 0$ **do**
        ▷ Find min remaining capacity on path p_{ab}
        $c := \min\{c(e), \frac{1}{r_p^a}\rho c(e^A), \frac{1}{r_p^b}\rho c(e^B)\}$,
          $p_{ab} \in SPT_a,\ e, e^A, e^B \in p_{ab}$;
        Route flow along p_{ab}
        ▷ Update edge length, capacity and cost
        $c^r(e) := c^r(e) - c,\ e \in p_{ab}$
        $t_{ab}^r = t_{ab}^r - c$
        $l(e) = (1 + \epsilon c/c(e))l(e)$
        $l(e^A) = (1 + \epsilon r_p^a c/\rho c(e^A))l(e^A)$
        $l(e^B) = (1 + \epsilon r_p^b c/\rho c(e^B))l(e^B)$
        $\phi := (1 + \epsilon c\eta_p^a/T_a M + \epsilon c\eta_p^b/T_b M)\phi$
      end for
    end while
  end for
end while
Scale down the flow to obtain feasible solution.

The a^{th} iteration of each phase of the algorithm considers traffic from virtual node a and terminates when all traffic from that node has been routed. At each step of the a^{th} iteration, we build a Dijkstra shortest path tree rooted on vertex a of the augmented graph. Flow is routed along the shortest path, and the amount of flow routed between each pair of virtual nodes is equivalent to the minimum remaining edge capacity along this shortest path. In particular, the flow we put on the augmented edge that represents the substrate node is r_p^a times the flow we put elsewhere. In terms of minimum capacity along the path, capacity of the augmenting edge is factorized by $1/r_p^a$ to ensure that the substrate node is not overloaded.

Each time we route flow along a particular edge, the remaining capacity of that edge is updated accordingly, along with the length function $l(e)$ and ϕ. If the remaining flow is less than the capacity of the edge, we route all remaining flow along the shortest path. The algorithm will terminate when the weighted sum in (3.26) $D(l, \phi) \geq 1$.

Note that the total amount of flow we route will violate the capacity constraints. To obtain a feasible solution, we determine the most congested links and we also compute the total number of migrated nodes from $x(p)$. Let Δ be the maximum ratio by which any of the constraints are violated. Then, we scale down the all the flows by a factor of Δ to obtain a feasible solution.

Using proof techniques similar to the ones in [36], we can show that this algorithm obtains a solution that is within $1+\omega$ of the optimal one, and its runtime is $\tilde{O}(\omega^{-2}|E|^2)$.

Proof. The FTPAS algorithm to solve the MCMCF problem consists of phases, iteration, and steps. Assume on step s of i^{th} phase, a^{th} iteration, we route flow from the a^{th} virtual node across the network. We denote the length of the substrate links and augmenting links as $l(e)$ and $l(e^A)$ respectively. Also, we denote the penalty factor to present the cost of link as ϕ.

Base on how we route the flow across the network, the following equations would hold:

$$\begin{aligned} l_{i,a}^{s}(e) &= l_{i,a}^{s-1}(e)[1 + \epsilon \frac{total\ new\ flow\ through\ e}{c(e)}] \\ &= l_{i,a}^{s-1}(e)[1 + \epsilon \frac{\sum_b f_{i,a,s-1}^{a \to b}}{c(e)}] \end{aligned} \tag{3.30}$$

The $f_{i,a,s-1}^{j \to b}(e)$ stands for the flow we route on the link e along path p during this step.

Similarly, for augmenting link:

$$l_{i,a}^{s}(e^A) = l_{i,a}^{s-1}(e^A)[1 + \epsilon \frac{\sum_b r_p^a f_{i,a,s-1}^{a\to b}}{c(e^A)}] \tag{3.31}$$

For the penalty factor:

$$\phi_{i,a}^{s} = \phi_{i,a}^{s-1}\{1 + \frac{\epsilon \sum_b f_{i,a,s-1}^{a\to b}}{N}(\frac{\eta_p^a}{T_a} + \frac{\eta_p^b}{T_b})\} \tag{3.32}$$

One observation is that the following inequality would hold for $l_{i,a}^{s}(e)$, $l_{i,a}(e^A)$ and $\phi_{i,a}^{s}$ respectively:

$$\begin{aligned} l_{i,a}^{s}(e) &\le (1+\epsilon) l_{i,a}^{s-1}(e) \\ l_{i,a}^{s}(e^A) &\le (1+\epsilon) l_{i,a}^{s-1}(e^A) \\ \phi_{i,a}^{s} &\le (1+\epsilon)\phi_{i,a}^{s-1} \end{aligned} \tag{3.33}$$

The first two equations can be easily verified, as in each step, we do not allow more flow to be routed on the substrate links and augmenting links than its capacity. As for the third inequality, because the total flow we route will be no greater than the total flow requirement T_a, we know that $\sum_b f_{i,a,s-1}^{a\to b} \le T_b$. Based on our two assumptions in 3.3, we know that $T_b \ge T_b$, and $N \ge 2$. Thus,

$$\phi_{i,a}^{s} \le \phi_{i,a}^{s-1}\{1 + \frac{2\epsilon}{N}\} \le (1+\epsilon)\phi_{i,a}^{s-1}$$

Using the equations (3.31) to (3.33), relationships can be established for $D(l_{i,a}^{s}, \phi_{i,a}^{s})$ and $D(l_{i,a}^{s-1}, \phi_{i,a}^{s-1})$:

$$\begin{aligned} D(l_{i,a}^{s}, \phi_{i,a}^{s}) &= \sum c(e) l_{i,a}^{s}(e) + \sum c(e^A) l_{i,a}^{s}(e^A) + \sigma N \phi_{i,a}^{s-1} \\ &= D(l_{i,a}^{s-1}, \phi_{i,a}^{s-1}) + \epsilon \sum_e \sum_b f_{i,a,s-1}^{a\to b} l_{i,a}^{s-1}(e) \\ &\quad + \epsilon \sum_{e^A} \sum_b r_p^a f_{i,a,s-1}^{a\to b} l_{i,a}^{s-1}(e^A) \\ &\quad + \epsilon\{\sum_b f_{i,a,s-1}^{a\to b}(\frac{\eta_p^a}{T_a} + \frac{\eta_p^b}{T_b})\}\phi_{i,a}^{s-1} \end{aligned} \tag{3.34}$$

By changing the order of summation:

$$\sum_{e}\sum_{b} f_{i,a,s-1}^{a\to b} l_{i,a}^{s-1}(e) = \sum_{b} f_{i,a,s-1}^{a\to b} \sum_{e} l_{i,a}^{s-1}(e)$$
$$\sum_{e^A}\sum_{b} r_p^a f_{i,a,s-1}^{a\to b} l_{i,a}^{s-1}(e^A) = \sum_{b} f_{i,a,s-1}^{a\to b} \sum_{e^A} r_p^a l_{i,a}^{s-1}(e^A) \tag{3.35}$$

We shall obtain:

$$D(l_{i,a}^s, \phi_{i,a}^s) = D(l_{i,a}^{s-1}, \phi_{i,a}^{s-1}) + \epsilon \sum_{b} f_{i,a,s-1}^{a\to b} \{\sum_{e} l_{i,a}^{s-1}(e)$$
$$+ r_p^a l_{i,a}^{s-1}(e^A) + r_{e^{m'}} l_{i,a}^{s-1}(e^{m'}) + (\frac{\eta_p^a}{T_a} + \frac{\eta_p^b}{T_b})\phi_{i,a}^{s-1}\}$$

Observe that the last part is the shortest path length between node a and b, this will give us:

$$D(l_{i,a}^s, \phi_{i,a}^s) = D(l_{i,a}^{s-1}, \phi_{i,a}^{s-1}) + \epsilon \sum_{b} f_{i,a,s-1}^{a\to b} dist_{ab}(l_{i,a}^{s-1}, \phi_{i,a}^{s-1}) \tag{3.36}$$

As $l_{i,a}^s$ is non-decreasing function, we can obtain that

$$D(l_{i,1}^s) \leq D(l_{i-1,1}^s) + \epsilon\alpha(l_{i,1}^0) \tag{3.37}$$

Based on the set of inequalities in (3.33) and (3.37), the rest of the proof will the same in [36], which would show that this algorithm would solve the LP problem in $\tilde{O}(\omega^{-2}m^2)$ time with $1+\omega$ optimal ratio. □

Bisection Search for σ

Recall that in the MCMCF formulation, we must set parameter $\sigma = \lambda^\star$ in order to solve the problem and obtain the optimal assignment of routes. Since we do not know $\lambda^\star$ in advance, we first establish lower and upper bounds for σ and then carry out a binary search to find a value for σ that yields a solution close to $\lambda^\star$.

As reconfiguration shall yield a solution no-worse than the current configurations, its throughput shall serve as lower-bound for σ. The upper-bound of the throughput is when the load on substrate links or nodes is completely balanced, in another word, we take the minimum ratio of $\{link\ resource/link\ request, node\ resource/node\ request\}$ as the

Algorithm 6 Overall Algorithm for MCMCF

Input:

σ^L: σ upper bound σ^U: σ lower bound

Output:

Collection of nodes to be migrated.

1: Construct the augmented graph G^a
2: Find lower and upper-bound, σ^L and σ^U
3: **while** $\sigma \neq \lambda^E$ **do**
4: setting $\sigma^E = (\sigma^L + \sigma^U)/2$
5: $x(p), \lambda(\sigma^E) \leftarrow$ solve MCMCF by FPTAS
6: **if** $\sigma^E > \lambda(\sigma^E)$: $\sigma^U = \lambda(\sigma^E)$, **else**: $\sigma^L = \lambda(\sigma^E)$
7: **end while**
8: Obtain the migration indicator from $x(p)$ from (3.18)
9: Select virtual nodes with largest migration indicator

given upper bound for the throughput value.

With the upper and lower bound established, we can then carry out the bisection search for a suitable σ. We denote the upper and lower-bound for σ as σ^U and σ^L respectively. We use the mid-point of the interval, $(\sigma^L + \sigma^U)/2$ as estimated σ^E. The bisection will terminate if the throughput achievable via FPTAS, $\lambda(\sigma^E)$, is close to σ^E, otherwise, we shall update σ^L and σ^U accordingly and iteratively repeat the process until this we find σ.

Complete Algorithm

The algorithm for virtual nodes selection is shown as Algorithm 2. We first construct augmented graph that includes the SN and VNs, and then we iteratively solve the MCMCF problem with a given parameter σ. We terminate this process once we identify an appropriate value for σ.

After normalization, we obtain the migration x_A^a based on the assignment of flow in the primal problem according to equation (3.18), and we use this value of the indicator for virtual node migration. Then, we select M virtual nodes with the largest migration indicator, mark them for reconfiguration, and pass them to the second subproblem that we discuss next.

3.3.2 Remapping of Selected Virtual Nodes

Inspired by the work of [29] and [15], we propose an algorithm to map the virtual nodes selected by the previous subproblem onto new substrate nodes, based on random walk on a Markov chain. For each virtual node a to be migrated, the algorithm assigns a numerical value to each substrate node A that represents the fitness of mapping a to A. We use this fitness value to reduce the search space for placing a virtual node. Specifically, for each virtual node, we select the N_s substrate nodes with the highest fitness value, and then search for the most suitable substrate node using an MIP.

The algorithm to assign a fitness value to the substrate nodes is presented as Algorithm 3. First, we compute the remaining link capacity $c^r(e)$ after removing the virtual nodes to be migrated. For each virtual node to be migrated, we compute the traffic between this node and all the substrate nodes. For example, if node a is selected for migration, and it communicates with nodes b and d, both of whom are mapped on substrate node A, the amount of traffic between a and A is $t_{ab} + t_{ad}$. Based on this value, we generate a probability vector for virtual node a that represents the initial probability of mapping a to each substrate node A:

$$\pi^{(0)}_{a\to A} = \frac{\sum_{b:b\to A} t_{ab}}{\sum_b t_{ab}} \tag{3.38}$$

We also generate the transition probability matrix by letting the transition probability from A to B as:

$$tp_{AB} = \frac{c^r(e_{AB})}{\sum_D c^r(e_{AD})} \tag{3.39}$$

At each step, the probability $\pi^{(t+1)}_{a\to A}$ is updated by the follows expression:

$$\pi^{(t+1)}_{a\to A} = \mu\pi^{(t)}_{a\to A} + (1-\mu)\sum_B \gamma^t_{AB} tp_{BA}\pi^{(t)}_{a\to B} \tag{3.40}$$

where μ is a smoothing factor which controls the rate of transition. The term γ^t_{AB} is defined as

$$\gamma^t_{a,B} = \pi^{(t)}_{a\to B} / \sum_a \pi^{(t)}_{a\to B} \tag{3.41}$$

and captures the interference between the placement of different virtual nodes on the substrate link.

Algorithm 7 Selection of Candidate Substrate Nodes

Input: $c^r(e)$: remaining substrate link capacity
t_{ab}: traffic between virtual node a and b
T_M: number of iterations for random walk
N_s: number of candidate substrate node to be selected

Output:
Set of candidate substrate nodes for each virtual node

1: For each virtual node a, construct initial probability distribution on according to equation (3.38)
2: Construct transition probability with equation (3.39)
3: **for** $j = 1, 2, \cdots, T_M$ **do**
4: Update probability distribution according to (3.40)
5: **end for**
6: For each virtual node i, select N_s substrate network with largest probability distribution π

The algorithm terminates after a T_M steps, at which time, for each virtual node a to be migrated, we select the N_S substrate nodes with the largest π value as the candidate substrate nodes for migration.

The operation of this algorithm can be interpreted as follows. The initial probability values are determined by the intensity of traffic between a virtual node and a substrate node, such that a substrate node that receives more traffic from a virtual node will have higher weight in the remapping process. With this initial probability distribution, we start a random walk on the Markov chain. During the t^{th} transition step, aside from $\pi^{(t-1)}$, the probability distribution from the previous step, two additional factors affect the transition rate: the remaining link capacity, and parameter μ. Specifically, the transition rate is proportional to the remaining link capacity, meaning that virtual nodes are more likely to be placed onto substrate nodes with a larger amount of available link capacity towards the destination. Also, the transition rate is affected by the value for other virtual nodes, i.e., if a link is likely to be shared among several virtual links, the available network resource decreases accordingly, and the probability of sharing also decreases. We note that we have the algorithm terminate after a fixed number of steps, instead of when the probability vector converges. This helps with increasing locality in the mapping, as it tends to place a virtual node upon a substrate node close to other substrate nodes it communicates with, which in turn helps reduce the overall traffic.

In summary, by constructing a Markov chain in the above manner, and carrying out a random walk for a fixed number of iterations, we obtain a probability vector that represents the likelihood of placing the selected virtual nodes onto substrate nodes. The probability reflects the fitness of placement by taking into account available bandwidth, interference with other nodes, and locality of communication. For each virtual node, we select N_s substrate nodes as candidates.

As a final step, given the above probability vector, we determine a new mapping of the selected virtual nodes onto the substrate network by solving an MIP which aims at minimizing jointly the node utilization, link utilization, and cost of flow. This MIP can be efficiently solved by setting N_s to a small number. The formulation, in essence, is similar to MIP formulation in Chapter III.

3.4 Evaluation

We consider three different schemes in our evaluation study:

1. no reconfiguration (no-rcnfg),
2. reconfiguration of virtual links only (l-rcnfg) using the algorithm in [61], and
3. reconfiguration of both virtual links and nodes (ln-rcnfg(M)) with our proposed algorithm, where M stands for the maximum number of nodes that may be migrated during each iteration; we use $M = 2, 4, 6$ in this study.

We compare the three schemes on three performance metrics: maximum (substrate) link utilization, maximum (substrate) node utilization, and InP revenue, defined as the sum, over all requests, of the bandwidth and node resource request of a request times the request's lifetime.

For the experiments, we developed an embedding testbed similar to the one in [61]. The topology of the substrate network and the virtual network requests are generated by the GT-ITM modeling tool [1] The substrate network consists of 50 nodes. The capacity of the substrate links and nodes follow a uniform distribution in $[50, 100]$. The number of virtual nodes in each request is an integer uniformly distributed in $[2, 10]$, while the requirements of virtual nodes and links are also uniformly distributed in $[20, 40]$.

The arrival and departure intervals of the virtual requests follow an exponential distribution, with a mean inter-arrival time of 10 and a mean inter-departure time of 100.

For each simulation, we generate 1000 virtual requests, and embed each arriving virtual using the same virtual network embedding algorithm of [61]. We assume that reconfiguration takes place periodically at constant intervals, and we vary the interval between reconfiguration events. And for each VM to be migrated, we select $N_s = 10$ substrate nodes as candidate substrate nodes.

Figure 3.3 plots the maximum link utilization under three different cases. Compared to the case of no reconfiguration (non-rcnfg), our algorithm (ln-rcnfg) reduces link utilization by at least 28%, and further as the maximum number M of nodes to be migrated increases. Compared to reconfiguration of virtual links only (l-rcnfg), our scheme decreases the maximum link utilization up to 15% when $M = 6$. Figure 3.4 plots the maximum node utilization. Again, our algorithm improves upon the no- or link only-reconfiguration by up to 22%, but the results show that the improvement is sensitive to the value of M, which must be selected appropriately.

Finally, the revenue of the infrastructure provider is shown in Figure 3.5. As we can see, link only reconfiguration results in higher revenue than no reconfiguration, as expected, as it may accommodate additional requests. Our algorithm provides further increase in revenue, between 12-36% compared to link-only reconfiguration, and from 17-50% compared to no reconfiguration.

We note that revenue and resource utilization are two seemingly conflicting goals, since an increase in revenue often implies higher resource utilization as more demands are mapped onto the physical infrastructure. An important insight that we gain from the above set of results, is that a tradeoff between utilization and revenue does not necessarilly exist. Migrating a small fraction of virtual nodes will not only help accommodate additional virtual network requests, but will also drive down resource utilization. In other words, by using intelligent algorithms such as the ones we presented in this work, the InP may benefit from an increase in revenue while also providing customer with better service.

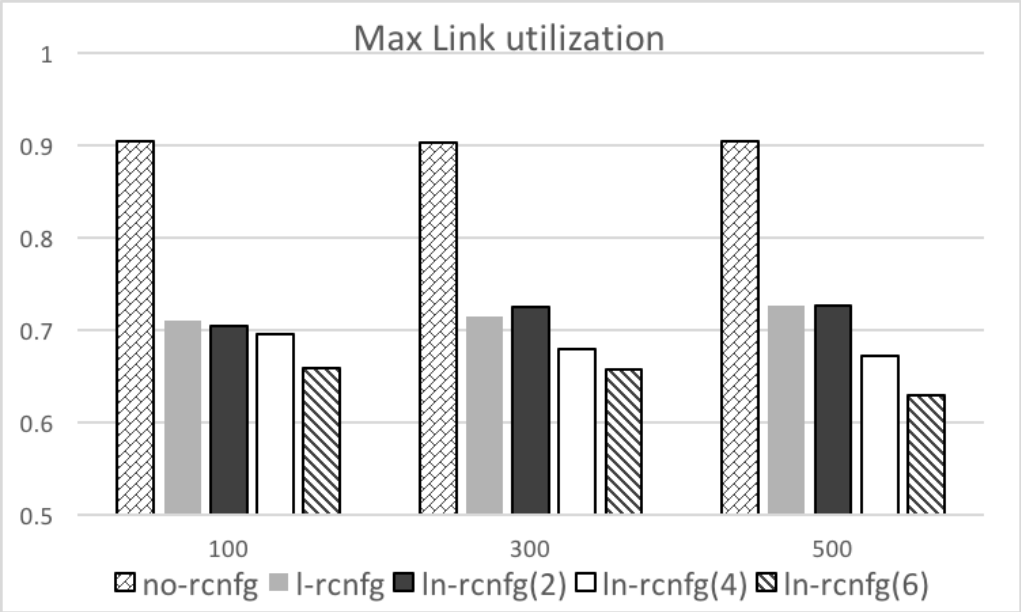

Figure 3.3: Maximum link utilization vs. number of reconfiguration events

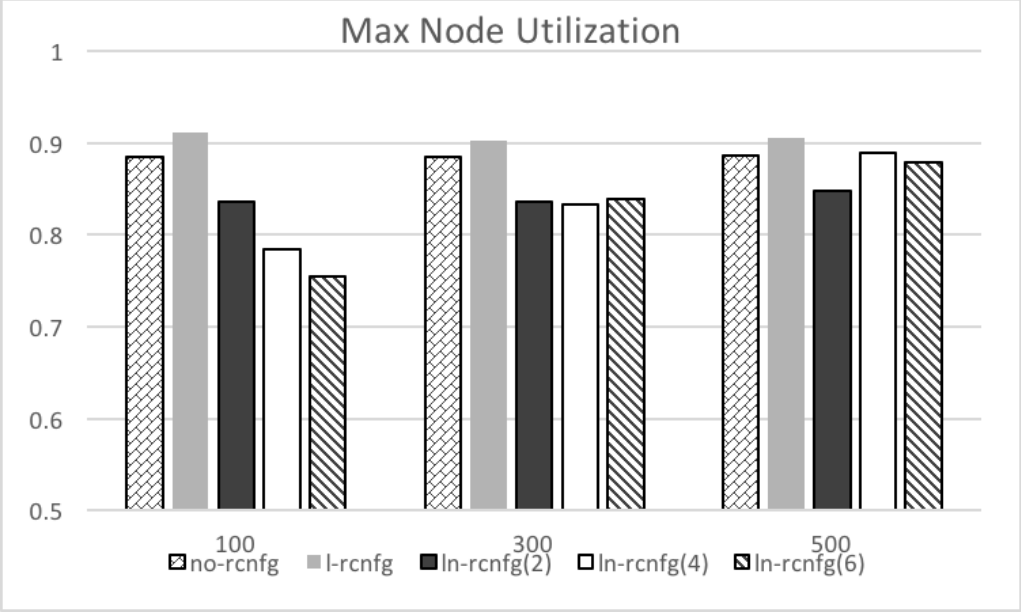

Figure 3.4: Maximum node utilization vs. number of reconfiguration events

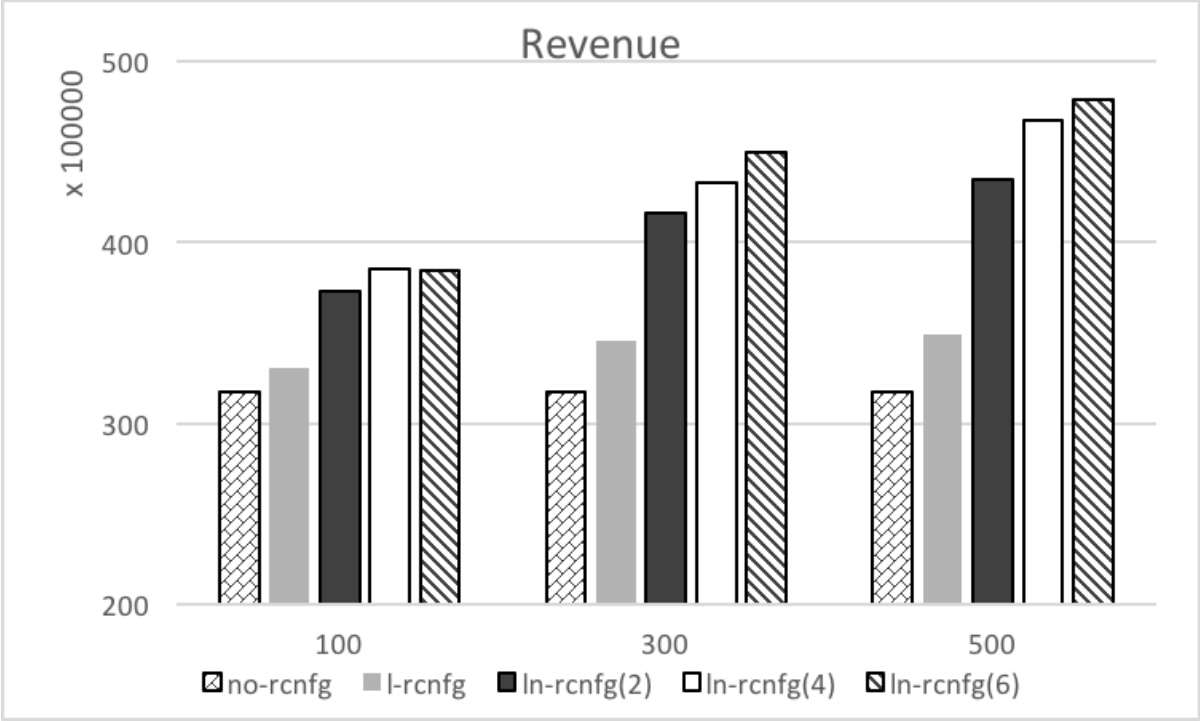

Figure 3.5: InP revenue vs. number of reconfiguration events

Chapter 4

Service Chain Routing in Network Function Virtualization

In this chapter, we focus on the service chain routing in an online scenario, and we present the algorithm to find a feasible routing of the service chain with respect to the service ordering constraints. The objective is to improve network performance by minimizing the maximum congestion. Under a reasonable assumption that is satisfied in practice, i.e., that congestion does not result from a single request, we prove that this algorithm is $O(\log m)$-competitive, where m is the number of edges in the network graph. We further show that this competitive ratio is asymptotically optimal.

In Chapter 4.1 we review the literature in this field. In Chapter 4.2, we present the model for the network and service requests, and formally define the service chain routing problem we consider. In Chapter 4.3, we present an online algorithm to minimize the maximum congestion and derive its competitive ratio.

4.1 Related Work for Service Chain Routing Problem

In recent years, research efforts have been directed towards the service chain embedding problem, and a survey of resource allocation problems in NFV was presented in [32]. A majority of these works [40,42,57,59] addressed an offline problem, where the aggregated requests need to be mapped onto the underlying network in one shot, under various em-

bedding objectives and scenarios. In [40,59], the authors considered the throughput maximization problem and proposed a randomized algorithm with performance guarantees, with applications to inter-datacenter networks and cellular networks. In [57], the authors considered the placement of VNFs and routing the traffic with a heuristic algorithm so as to minimize the expensive optical/electrical/optical conversions in datacenters. In [42], a heuristic algorithm based on game theory was proposed to place the virtual network functions and route the traffic to minimize operational cost.

Apart from the works that addressed the offline service chain routing problem, there also exist several studies that consider the online case. In [10], a service chain orchestration routing strategy was proposed to route the traffic so as to satisfy the service ordering constraints. However, the underlying link load and capacity were not taken into account in that work. In [24], the authors presented a multipath routing algorithm for online service provisioning, which was obtained by solving a linear programming problem, while in [60], the authors proposed an admission control scheme to admit and route online requests. In this work, on the other hand, we consider the online problem of routing unicast traffic along a single path.

The two studies most related to our work are [21, 43]. The objective of both is to map incoming network service requests to the physical network with finite capacity, and jointly consider the service chain embedding problem with admission control. In [43], the authors proposed an online algorithm that maximizes the number of admitted requests, under node capacity constraints, and has an $O(\log K)$ competitive ratio, where K is the number of network functions in the service chain. A "standby" mode was introduced in [21] to defer the acceptance of a request when sufficient resources are not available. Under this architecture, the authors proposed an online algorithm for the service chain embedding problem with the objective of maximizing the revenue with link capacity considerations. In our work, we tackle the problem from a different perspective, i.e., we assume that all services are admitted and our objective is to embed the service chain in a way that minimizes the maximum congestion.

4.2 Network Model and Problem Formulations

4.2.1 Network Model

We model the network as an undirected graph $G = (V, E)$, with $n = |V|$ number of vertices, and $m = |E|$ number of edges. The edges are capacitated, with $c_{u,v}$ denoting the capacity of edge $(u, v) \in E$. For the ease of presentation, when the two endpoints of an edge are irrelevant, we denote the edge as $e \in E$, and its capacity as $c(e)$. The network supports a set of L distinct network functions, $NF = \{NF_1, NF_2, \ldots, NF_L\}$, and each network function NF_l is deployed (instantiated) at a subset $V_l \subseteq V$ of the network nodes. In addition, each network node may support an arbitrary number of the NFs. We assume that the placement of NFs on network nodes (i.e., the sets $V_l, l = 1, \ldots, L$) is provided as input to the problem.

4.2.2 Service Chain Request

We model the service chain request $\mathcal{C}_i$ as a tuple $\mathcal{C}_i = (src_i, dst_i, d_i(k), \mathcal{F}_i)$, where src_i and dst_i are the source and destination nodes for the service chain and $\mathcal{F}_i = \left\{f_i^{(1)}, f_i^{(2)}, \ldots, f_i^{(k_i)}\right\}$, $f_i^{(k)} \in NF$, is the set of network functions that the traffic of this request must traverse in the given order. We use k_i to represent the number of NFs in the service chain $\mathcal{C}_i$. Similar to the work in [44], we assume that some network functions (e.g., an encoder or WAN optimizer) may have traffic changing effects, such that the amount of traffic coming out of the network function be different than the amount of traffic that went in. As a result, the amount of traffic on each segment of the path may vary, and we use $d_i(k)$ to represent the amount of traffic on the k-th segment.

In this work, we assume that requests are permanent, i.e., once a request arrives it will never terminate; extending the algorithm to the scenario whereby requests have a certain holding time after which they release resources and leave the network is the subject of ongoing research.

4.2.3 Problem Formulation

Service requests are routed in an online fashion, such that each request is routed without any information about the arrival time, traffic volume, or network functions of the future requests. The service chain will be routed in a way that each packet passes through the

VNFs in a predefined order. We define the decision variable $f_{u,v}^{i,k}$ to represent if the k^{th} segment of $\mathcal{C}_i$ is routed on the edge (u, v). The objective is to route a new request $\mathcal{C}_i$ so as to minimize the maximum network congestion. We define the congestion metric after we route the request $\mathcal{C}_i$ as $U_i = max_{(u,v)} \sum_{i,k} f_{u,v}^{i,k} d_i(k)/c_{u,v}$

Based on the above definitions, we formulate the online congestion minimization problem as the following integer linear programming problem (ILP):

$$minimize \quad U_R \tag{4.1}$$

$s.t.$

$$\sum_{j \in \delta(src)} f_{src,j}^{i,0} - \sum_{j \in \delta(src)} f_{j,src}^{i,0} = 1 \tag{4.2}$$

$$\sum_{j \in \delta(dst)} f_{j,dst}^{i,k_i} - \sum_{j \in \delta(dst)} f_{dst,j}^{i,k_i} = -1 \tag{4.3}$$

$$\sum_{v \in \mathcal{F}_k} \sum_{u \in \delta(v)} f_{u,v}^{i,k} - \sum_{v \in \mathcal{F}_k} \sum_{u \in \delta(v)} f_{v,u}^{i,k} = 1, 1 \leq k < k_i \tag{4.4}$$

$$\sum_{u \in \delta(v)} f_{u,v}^{i,k} - \sum_{u \in \delta(v)} f_{v,u}^{i,k} = \sum_{u \in \delta(v)} f_{u,v}^{i,k+1} - \sum_{u \in \delta(v)} f_{v,u}^{i,k+1},$$

$$0 \leq k < k_i, v \in \mathcal{F}_k \tag{4.5}$$

$$\sum_{u \in \delta(v)} f_{u,v}^{i,k} - \sum_{u \in \delta(u)} f_{v,u}^{i,k} = 0, v \notin \mathcal{F}_k, 1 \leq k < k_i \tag{4.6}$$

$$\sum_{i,k,u,v} f_{u,v}^{i,k} \leq U_R c_{u,v} \tag{4.7}$$

$$f_{u,v}^{i,k} = \{0, 1\} \quad \forall i \leq R, \ w \in W_i \tag{4.8}$$

Expression (4.1) represents the objective of minimizing the maximum congestion at the time request $\mathcal{C}_R$ is routed. As we are solving an online problem, the same objective must have been applied to all earlier requests $\mathcal{C}_i, i < R$.

Constraint (4.2) to constraint (4.6) are the flow conservation constraints: Constraint (4.2) and constraint (4.3) respectively guarantee that the traffic originates from the source node and directs towards the destination node. Constraint (4.4) ensures that, for the k^{th} segment of the service chain, the traffic will be routed towards one of the VNFs in $\mathcal{F}_k$. And the constraint (4.5) ensures that the flow in and out of a single VNF is consistent. Because the traffic will be processed by one of the VNFs in $\mathcal{F}_k$, when the traffic comes out

of this VNF, it will switch to the next segment. In another word, the destination VNF in the previous segment needs to be the source of the next segment. The constraint (4.4) and constraint (4.5) together ensure that the traffic will pass through the VNFs in a predefined order. Constraint (4.6) guarantees that the net flow in and out of a VNF remains 0. Those flow conservation constraints, combined with constraint (4.8) that enforces a binary value for the decision variable, ensures that all the traffic of one request will be routed along a single walk.

Lastly, (4.7) specifies that the total amount of traffic carried by any edge will not exceed the product of the edge capacity times U_R. Consequently, by minimizing U_R in the objective function we minimize the maximum edge congestion.

4.3 Online Routing Algorithm

In this section, we propose an online service chain routing algorithm. We design the algorithm under two different considerations: first, minimizing the maximum link utilization, in which it achieves an asymptotically optimal competitive ratio of $O(\log m)$; second, reducing the number of hops for routing so as to lower the delay and packet loss rate. The algorithm is inspired by the virtual circuit routing problem [4] and routes each incoming request along the shortest walk under a customized length function. In the following, we first define a set of concepts used in developing the algorithm, and then provide the algorithm details and evaluate its performance.

4.3.1 Definitions

Valid Walk. Observe that when a service chain $\mathcal{C}_i$ is routed on a physical network, each segment of the traffic is routed along a path. This renders the routing of $\mathcal{C}_i$ as a walk on the graph G. We use W_i to denote the set of all valid walks for request $\mathcal{C}_i$, i.e., the walks that start at node src_i, traverse the required set of network functions in the given order, and terminate at node dst_i. We denote the walk selected by the routing algorithm for service request $\mathcal{C}_i$ as $w_i \in W_i$. If the request is routed along the walk w_i, then the amount of traffic along each edge e of the walk may be determined from quantities $d_i(k)$ of the request tuple. This is illustrated in Figure 4.1, where the service request shown at the top of the figure requires one unit of traffic from node A to the network function NF and from NF to node D. The dotted line on the bottom of Figure 4.1 shows a walk that

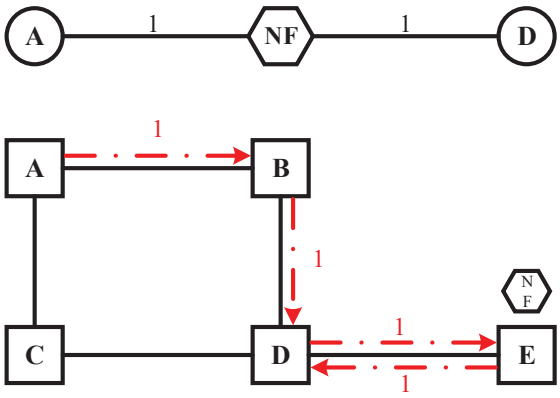

Figure 4.1: A service chain request (top) and corresponding walk on the topology graph (bottom)

represents a valid embedding of this service chain on the network topology, assuming that the network function is located at node E. From this walk, we determine that one unit of traffic goes through the edges (A, B) and (B, D), while two units of traffic are placed on edge (D, E). We use $tr_i(e, w)$ to denote the amount of traffic from request $\mathcal{C}_i$ that travels along edge e of walk w.

Penalty function. The penalty function serves as an indicator for the congestion of an edge e. For each edge e, we define the penalty function after we route request $\mathcal{C}_i$ as:

$$p_i(e) = \gamma^{\frac{l_i(e)}{c(e)L^*}}, \tag{4.9}$$

where L^* is the optimal congestion of the network in hindsight, γ is a constant (more details on L^* and γ shortly), and $l_i(e)$ is the load on edge e after we route the first i requests, namely, $l_i(e) = \sum_i tr_i(e, w_i)$.

Potential function. We use the potential function $\phi(i)$, defined as the sum of the edge penalty functions after we route the request $\mathcal{C}_i$,

$$\phi(i) = \sum_{e \in E} p_i(e), \tag{4.10}$$

to capture the overall cost of placing the first i requests.

Shortest path tour. The shortest path tour problem (SPTP) [25, 26] has been studied extensively in the literature in various contexts. The input to the problem is a

weighted graph G, source src and destination dst nodes, and multiple subsets of nodes $\{T_1, T_2, \ldots, T_K\}$. An algorithm for SPTP finds a walk (i.e., one or more edges may be traversed multiple times as part of the walk) from src to dst that visits at least one of the nodes in each set $T_k, 1 \leq k \leq K$, sequentially. Additionally, the algorithm must construct a walk whose weighted length is shortest among all valid walks.

We note that, assuming subsets T_k represent the sets of nodes V_l where instances of each network function NF_l are placed, then an (online) algorithm for SPTP will find a walk for routing an incoming service chain request. Therefore, our goal is to define a length function $len(e)$ for each edge such that the online SPTP algorithm will have a low competitive ratio with respect to congestion minimization. To this end, we first examine how congestion minimization is related to the potential function, and in turn how it translates into an appropriate length function $len(e)$ for SPTP.

Our first observation is that the log value of the potential function, $\log(\phi(i))$, is an upper bound for the competitive ratio. Specifically, the potential function is the sum of all penalties, and therefore greater than any single penalty value. More formally:

$$\phi(i) = \sum_{e \in E} p_i(e) \geq \max_e p_i(e) \tag{4.11}$$

$$= \max_e \gamma^{\sum_i tr_i(e,w_i)/c(e)L^*}. \tag{4.12}$$

By taking the logarithm of both sides, and given the earlier definitions of γ and L^*, it follows that the competitive ratio is bounded by $\log(\phi(i))$.

Furthermore, notice that the value of $\phi(i-1)$ is constant since the routing of all previous service requests is given (i.e., it is fixed and may not change). Therefore, minimizing the increment of the potential function, $\phi(i) - \phi(i-1)$, is inherently equivalent to minimizing the potential function $\phi(i)$ for this online problem. Now, we also observe that when service request $\mathcal{C}_i$ is routed along the walk w_i, only the penalty function for the edges $e \in w_i$ will change. Thus, the increment to the potential function can be rewritten as:

$$\begin{aligned} \phi(i) - \phi(i-1) &= \sum_{(e) \in w_i} (p_i(e) - p_{i-1}(e)) \\ &= \sum_{e \in w_i} \gamma^{\frac{l_{i-1}(e)}{c(e)L^*}} (\gamma^{\frac{tr_i(e,w_i)}{c(e)L^*}} - 1) \end{aligned} \tag{4.13}$$

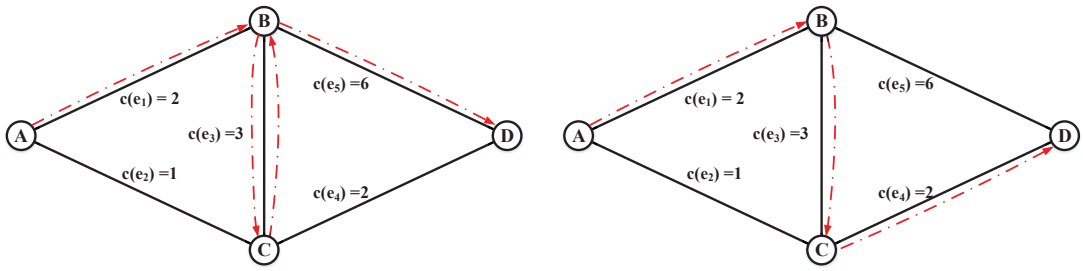

Figure 4.2: Walks under the two length functions for the example of Chapter 4.3.1.

We conclude that minimization of the potential function reduces to finding the shortest valid walk on G whose length is defined by (4.13).

However, the penalty associated with walk w depends on both the number of times and the order in which w traverses an edge e. This is a crucial difference with SPTP, as the traffic is in the exponent of the second factor in (4.13), i.e., $\gamma^{\frac{tr_i(e,w)}{c(e)L^*}}$. In other words, if w traverses e multiple times, we cannot simply add the cost to obtain the penalty of the walk. This raises the question of what value to assign to each edge if we are to use an algorithm for SPTP to find the walk. In particular, we face a crucial dilemma: apparently, $tr_i(e,w)$, the amount of traffic we put on e, depends on the actual walk w, but we do not have knowledge of the walk until we find out the route.

Length function. To address the above issue, we must define the length function so that it is independent of the walk selected for a service request. Specifically, we define the length function for edge e that is part of the walk for request $\mathcal{C}_i$ as:

$$len_i(e,k) = \gamma^{\frac{l_{i-1}(e)}{c(e)L^*}}(\gamma^{\frac{d_i(k)}{c(e)L^*}} - 1) \tag{4.14}$$

With this definition, the length function depends on the number k of segments of the walk (a value that is provided as input to the problem), as the amount of the traffic will change on different segments, but not on the specific walk selected.

Nevertheless, the length function (4.14) introduces another challenge. Specifically, a solution to SPTP (i.e., the shortest walk) using (4.14) as the length function may not be optimal under the function (4.13), since the length of the walk is inherently different. An example is shown in Figure 4.2, where we assume the new request is for one unit of traffic to be routed from source A to destination D after passing a network function located at node C and the capacity of each edge is as shown on Fig. 4.2. For this example, the

utilization $\frac{l_{i-1}(e)}{c(e)L^*}$ is assumed the same for all edges e prior to routing this new request, and also we assume $L^* = 1$. In this case, the solution to SPTP under length function (4.14) is the walk shown with a dotted line on the left side, while the optimal walk that minimizes the increase in potential function of (4.13) is different and shown in the right side.

Let us now assume that the maximum congestion cannot be the result of any one request, i.e., $\frac{\sum_k d_i(k)}{c(e)} \leq L^*, \ \forall i, \ e$. This is a reasonable assumption that is satisfied in practice in the common scenario that the capacity of each edge is large compared to the traffic demand of any single request. Under this assumption, we can state the following result regarding the walk selected by an SPTP algorithm with length function (4.14):

Lemma 1. *For a request $\mathcal{C}_i$, the shortest path tour w_i under the length function (4.14) is a γ-approximation to the optimal walk which minimizes (4.13).*

Before we move to the proof of the lemma ł:walk-length, we first define two length functions, namely, the *non-additive* length and *additive* length function for a walk. Additionally, we observe from (4.13), that the load on each edge is link-specific. For the sake of simplicity, we assume, without loss of generality, that all edges have capacity $c(e) = 1$.

We first denote as $len_{na}(e, w_i)$ the *non-additive* length function in (4.13), and as $len^w_{na}(w_i)$ the corresponding length of a walk under this non-additive edge length :

$$len_{na}(e, w_i) = \gamma^{\frac{l_{i-1}(e)}{L^*}}(\gamma^{\frac{tr_i(e,w_i)}{L^*}} - 1) \tag{4.15}$$

$$len^w_{na}(w_i) = \sum_{e \in w_i} len_{na}(e, w_i). \tag{4.16}$$

We also define an *additive* length for a walk:

$$len^w_a(w_i) = \sum_{k=0}^{k_i} \sum_{e \in s_k} len_i(e, k), \tag{4.17}$$

where the $len_i(e, k)$ is the length function defined in Chapter 4.3. The total contribution of edge e to the walk is given by: $len_a(e, w_i) = \sum_{k:e \in s_k} len_i(e, k)$.

In order to prove the Lemma 1, we first prove the following lemma.

Lemma 2. *For any walk w_i, the non-additive length of the walk is (a) bounded below by the additive length, and (b) bounded above by γ times the additive length, i.e.*

$$len^w_a(w_i) \leq len^w_{na}(w_i) \leq \gamma len^w_a(w_i) \tag{4.18}$$

Proof. We have the following observation: for a walk w_i, the ratio of the non-additive cost to the additive cost is bounded below and above by the minimum and maximum ratio of each edge, respectively:

$$\min_{e \in w_i} \frac{len_{na}(e, w_i)}{len_a(e, w_i)} \leq \frac{len^w_{na}(w_i)}{len^w_a(w_i)} \leq \max_{e \in w_i} \frac{len_{na}(e, w_i)}{len_a(e, w_i)} \tag{4.19}$$

The two length function is different when and only when w traverses e multiple times. As w_i stands for the routing of the request, the traffic placed on edge e is the same, regardless of the penalty function. Without loss of generality, we assume that the walk traverses the edge for the first k times. Then, the ratio of the two functions is:

$$\begin{aligned} \frac{len_{na}(e, w_i)}{len_a(e, w_i)} &= \frac{\gamma^{\frac{l_{i-1}(e)}{L^*}}(\gamma^{\frac{tr_i(e,w_i)}{L^*}} - 1)}{\sum_k \gamma^{\frac{l_{i-1}(e)}{L^*}}(\gamma^{\frac{t^k_i}{L^*}} - 1)} \\ &= \frac{\gamma^{\frac{\sum_k t^k_i}{L^*}} - 1}{\sum_k (\gamma^{\frac{t^k_i}{L^*}} - 1)} \end{aligned} \tag{4.20}$$

First, we prove that the non-additive cost is bounded below by the additive cost. Using the Taylor expansion of the exponential function, one may verify that $\gamma^{\sum_i x_i} - 1 \geq \sum_i (\gamma^{x_i} - 1)$ for all $\gamma \geq 1$ and $x_i \geq 0$, leading to the desired result:

$$\frac{len_{na}(e, w_i)}{len_a(e, w_i)} = \frac{\gamma^{\frac{\sum_k t^k_i}{L^*}} - 1}{\sum_k (\gamma^{\frac{d(k)}{L^*}} - 1)} \geq 1 \tag{4.21}$$

Next, we prove that the non-additive cost is bounded above by γ times the additive cost. The difference between the two costs on any edge e is:

$$\frac{len_{na}(e, w_i)}{len_a(e, w_i)} = \frac{\gamma^{\frac{\sum_k t^k_i}{L^*}} - 1}{\sum_k (\gamma^{\frac{t^k_i}{L^*}} - 1)} \leq \frac{(\gamma - 1)\frac{\sum_k t^k_i}{L^*}}{\sum_k (\gamma^{\frac{t^k_i}{L^*}} - 1)} \tag{4.22}$$

Inequality (4.22) holds due to two reasons: first, it is our assumption that a single request may not cause the most congestion, i.e., $\frac{\sum_k t^k_i}{L^*} \leq 1$; second, $\gamma^x - 1 \leq (\gamma - 1)x$, for $0 \leq x \leq 1$ and $\gamma \geq 1$.

Using the Maclaurin series $\gamma^x = \sum_m \frac{(ln\gamma)^m}{m!} x^m$, one may verify that $\gamma^x - 1 \geq \gamma x ln\gamma$

for $\gamma \geq 1$, leading to the following inequality:

$$\frac{len_{na}(e, w_i)}{len_a(e, w_i)} \leq \frac{(\gamma - 1)\frac{\sum_k t_i^k}{L^*}}{\sum_k (\gamma^{\frac{d(k)}{L^*}} - 1)} \leq \frac{(\gamma - 1)\frac{\sum_k t_i^k}{L^*}}{\sum_k ln\gamma \frac{d(k)}{L^*}} \tag{4.23}$$

$$= \frac{\gamma - 1}{\gamma ln\gamma}\gamma \leq \gamma \tag{4.24}$$

The last inequality (4.24) stems from the fact that $\frac{\gamma-1}{\gamma ln\gamma}$ is a monotonically decreasing function with respect to γ, and $\lim_{\gamma \leftarrow 1} \frac{\gamma-1}{\gamma ln\gamma} = 1$.

Combining (4.21) with (4.24) we obtain the stated lower and upper bounds for the non-additive length of any walk. □

We are now ready to prove Lemma 1.

Proof. Denote the shortest valid walk with respect to the additive length $len_a^w(w)$ as w^a, and the optimal walk with respect to the non-additive length $len_{na}^w(w)$ as w^*. Applying the lower and upper bounds of Lemma 4, we have the following two inequalities:

$$\frac{1}{\gamma} len_{na}^w(w^a) \leq len_a^w(w^a), \quad len_a^w(w^*) \leq len_{na}^w(w^*) \tag{4.25}$$

Since w^a is the shortest walk under the additive function, we have that:

$$len_a^w(w^a) \leq len_a^w(w^*) \tag{4.26}$$

Combining the last inequality with the two in (4.25), we obtain:

$$len_{na}^w(w^a) \leq \gamma \; len_{na}^w(w^*) \tag{4.27}$$

which proves Lemma 1. □

4.3.2 Online Algorithm

Before we describe our online algorithm, let us explain how to circumvent the fact that the value of L^*, the optimal congestion in hindsight, is unknown. Since we do not have any prior knowledge, we use λ as an estimate for L^*. Initially, we set λ set to be the minimum possible congestion for the first request, and we use this value in the length

function (instead of the unknown L^*). For request $\mathcal{C}_i$, after we map it using the initial value of λ, we examine all the edges. If there exists an edge e such that

$$tr_i(e, w_i) + l_{i-i}(e) \geq \log_\gamma(4m)\lambda, \ \forall e \in w_i \tag{4.28}$$

then we double the value of λ and remap the request $\mathcal{C}_i$. This approach does not affect the asymptotic competitive ratio. The proof to the correctness of this approach is in [4].

In addition, we describe below how to select a walk fewer hops. We recognize that a non-zero constant term in a length function would encourage the selection of a walk with fewer number of hops. If we switch the length function from (4.14) to

$$\begin{aligned} len_i^{'}(e,k) &= len_i(e,k) + \alpha * \rho \\ &= \gamma^{\frac{l_{i-1}(e)}{c(e)L^*}}(\gamma^{\frac{d_i(k)}{c(e)L^*}} - 1) + \alpha * \rho \end{aligned} \tag{4.29}$$

, where α and ρ are two positive constant (we shall discuss how to determine their values shortly) and run the SPTP algorithm, then we can expect a shorter walk than the one under (4.14). For the sake of brevity, we call the first term, $len_i(e,k)$, the *congestion term*, and the second, $\alpha * \rho$, the *constant term*. The reason we can expect a shorter walk is as follow. Suppose we have two different walks w_i and $w_i^{'}$, with the number of hops being $h(w_i^{'})$ and $h(w_i^{'})$, while $h(w_i) > h(w_i^{'})$. The constant term in a length functions respectively contribute $h(w_1) * \alpha * \rho$ and $h(w_2) * \alpha * \rho$ to the total length of the two walks. This would discourage the selection of the walk w_1 so long as the difference in the load term is not significant enough.

It remains to be a problem on how to select the constant term for each edge. We assign the value in the following way: first, given a service request, we route the service chain along an SPTP under the length function (4.14). This results in a walk $w_i^{'}$. The weighted length of $w_i^{'}$ is ω, and the number of hops is $h(w_i^{'})$. We select $\rho = \omega / h(w_i^{'})$ and have the following observation:

Lemma 3. *The shortest path tour under the length function (4.29) is* $(1+\alpha)\gamma$*-approximation to the optimal walk that minimizes (4.13).*

Proof. We denote the Shortest Path Tour under the length function (4.14) as $w^{'}$, and under length function (4.29) as w.

First, it is easy to verify that the weighted length of w under (4.29) is at most $(1+\alpha)\omega$. This is because the walk $w^{'}$ has the length of ω under the length function (4.14) and $(1+\alpha)\omega$ under (4.29). As $w^{'}$ is not necessarily the Shortest Path Tour, and we are running a Shortest Path Tour algorithm, the shortest walk we find under (4.29), namely, w, has a weighted length of at most $(1+\alpha)\omega$.

Second, notice that the constant term in length function (4.29) is positive, meaning that the length of w under (4.14) is strictly smaller than its length under (4.29). This implies that the length of $w^{'}$ is at most $(1+\alpha)\omega$ under length function (4.14). Following the result from Lemma 1, where ω is a γ-approximation solution to the weighted length, we can conclude the weighted length of w is at most $(1+\alpha)\gamma$-approximation to the optimal solution. □

With the proper concepts defined, our online algorithm is presented as Algorithm 1 below. We first build a graph using the length function (4.14), based on the link load from previous requests, the demand of the new request, and the estimate for the optimal congestion λ. Then, we route the request using a shortest path tour algorithm to find the walk $w_i^{'}$ based on this graph. We obtain the weighted length for this Shortest Path Tour ω, and the number of hops for this tour $h(w_i^{'})$. Using these two values, we re-compute the Shortest Path Tour under the length function (4.29), and obtain the tour $w_i^{'}$. For all edges $e \in w_i$ along this walk, we examine if the inequality (4.28) holds. If so, this suggests that our estimate of L^* is low; in this case, we double the value of the estimate λ, recompute the length function, and reroute the request $\mathcal{C}_i$. Otherwise, we route the request along walk w_i and update the load of the corresponding edges.

4.3.3 Performance Analysis

Time Complexity

Building a weighted graph can be completed in $O(m)$ time, hence the time complexity of Algorithm 1 is dominated by finding the shortest path tour, which can be completed in $(Km \log n)$ time [10], where K is the number of network functions in the request (i.e., the number of segments of the walk).

Due to the potential underestimation of L^*, the SPTP algorithm may have to be run multiple times for a single request. For each request, this will happen at most $\log_2(KDC)$ times, where D is a ratio of the maximum to minimum traffic demand, and C is the ratio

Algorithm 8 Online service chain routing

Input:

$l_{i-1}(e)$: existing load on edge e

C_i: new service chain to be routed.

λ: estimation for the optimal congestion in hindsight.

Output:

w_i: selected route for service chain i

1: Construct a weighted undirected graph G with edge length set according to (4.14), with λ in place of L^*.
2: Compute the shortest path tour $w_i^{'}$ to obtain the length ω and the number of hops $h(w_i^{'})$.
3: Re-compute the shortest path tour w_i with weight-length according to (4.29).
4: **if** $\exists e \in w_i,\ l_{i-1}(e) + tr_i(e, w_i) \geq \lambda \log_\gamma(4m)$ **then**
5: $\quad \lambda \leftarrow 2\lambda$, **goto** Step 1
6: **end if**
7: Update the load on each link.

of the maximum to minimum edge capacity. This follows from the fact that the minimum value that λ takes is $\lambda = \frac{\min_{i,k} t_i^k}{\max_e c(e)}$, while a maximum value in (4.28) is $\frac{\max_i \sum_k t_i^k}{\min_e c(e)}$. As the estimate doubles each time, the number of estimates is upper bounded by $O(\log(KDC))$.

Thus, the overall time complexity of this online algorithm is in $O(Km\ \log n\ \log(KDC))$.

Competitive Ratio

We now prove that our algorithm achieves a competitive ratio that is asymptotically optimal. In the first place, we make the following observation on the lower bound of the competitive ratio for Algorithm 1.

Lemma 4. *Service Chain Routing problem has a $O(\log m)$ lower bound for the competitive ratio.*

Proof. Observe that in the special case of $K = 0$, i.e., when the service does not request any network function between the source and destination nodes, the service chain routing problem reduces to the virtual circuit routing problem in [4]. The optimal competitive ratio of virtual circuit routing is $O(logm)$, which suggests that $O(logm)$ is the lower bound. □

We then show that the Algorithm 1 achieves a $O(logm)$ competitive ratio. Combined with Lemma 4, it suggests that the competitive ratio achieved by Algorithm 1 is asymptotically optimal.

To proof the competitive ratio of the Algorithm 1, we first prove that there is an upper bound for the potential function $\phi(i)$.

Lemma 5. *The potential function can be upper-bounded by $\phi(i) \leq \sigma m$, $\forall i$, where m is the number of edges of the network graph ans σ a constant number, with a proper choice of α and γ.*

Proof. Denote the shortest walk under (4.29) as w_i, and the optimal walk in hindsight as w_i^*.

Following the result of Lemma 2, the increase to the potential function is:

$$\phi(i) - \phi(i-1) = len_{na}^{w}(w_i) \leq (1+\alpha)\gamma \; len_{na}^{w}(w_i^*) \tag{4.30}$$

The sum of the differences above is given by:

$$\begin{aligned}
\phi(R) - \phi(0) = & \sum_{i=1}^{R} (\phi(i) - \phi(i-1)) && (4.31)\\
& \leq (1+\alpha)\gamma \sum_{i=0}^{R} len_{na}^{w}(w_i^*) && (4.32)\\
& = (1+\alpha)\gamma \sum_{i=0}^{R} \sum_{e \in C_i^*} \gamma^{\frac{l_{i-1}(e)}{L^*}} (\gamma^{\frac{\sum_k d_i(k)}{L^*}} - 1) && (4.33)\\
& \leq (1+\alpha)\gamma \sum_{i=0}^{R} \sum_{e \in C_i^*} \gamma^{l_R(e)} (\gamma - 1) \sum_k d_i(k)/L^* && (4.34)\\
& = (1+\alpha)\gamma(\gamma-1) \sum_{e \in C_i^*} \gamma^{l_R(e)} \sum_{i=0}^{R} \sum_k d_i(k)/L^* && (4.35)\\
& \leq (1+\alpha)\gamma(\gamma-1) \sum_{e \in C_i^*} \gamma^{l_R(e)} && (4.36)\\
& = (1+\alpha)\gamma(\gamma-1)\phi(R) && (4.37)
\end{aligned}$$

Inequality (4.34) results from the fact that $l_i(e)$ is non-decreasing with respect to i, and $\gamma^x - 1 \leq (\gamma - 1)x$. Inequality (4.37) holds as $L^* \leq \sum_{i=0}^{R} \sum_k d_i(k, j)$ is the optimal

congestion, in hindsight, for e for the first R requests.

By setting $\gamma = 1.4$ and $\alpha = 0.5$, we have the desired result: $\phi(R) < 6.25\phi(0) = 6.25|E| = 6.25m$.

□

Theorem 6. *The competitive ratio of Algorithm 1 is $O(\log m)$, which is asymptotically optimal for congestion minimization.*

Proof. First, we show that $O(\log m)$ is a lower bound on the competitive ratio. Observe that in the special case of $K = 0$, i.e., when the service does not request any network function between the source and destination nodes, the service chain routing problem reduces to the virtual circuit routing problem in [4], the optimal competitive ratio of which is $O(\log m)$. This suggests that $O(\log m)$ is the asymptotically optimal competitive ratio for Algorithm 1.

Next, we show that the competitive ratio achieved by our algorithm is $O(\log m)$. Combining inequality (4.12) and Lemma 2 and taking the logarithm on both sides, we obtain

$$max_{e \in E} \frac{l_i(e)}{c(e)L^*} \leq \log_\gamma(\sigma m) \tag{4.38}$$

which shows that the algorithm is $\log m$-competitive. □

4.4 Numerical Results

In this section, we present the numerical result to evaluate the performance of the online service chain routing algorithm.

4.4.1 Baseline algorithms

We use the following two methods as the baseline to our proposed algorithm (*competitive-online*):

- **Lp-Offline**: The first approach is via solving the LP problem in hindsight. In an offline scenario, we have all the information about the service chain in advance. We relax the integral constraint (4.8) for the ILP in section 4.2, and solve the corresponding LP-problem. The solution to the LP-problem will serve as the lower bound for maximal utilization minimization problem. In an offline case, we have

more information than an online scenario and the solution to the LP problem may be fractional. It would mean that the *Lp-Offline* will only provide a better load balancing than any online algorithm who steers the traffic along a single-walk. Thus, the gap between the proposed algorithm, *competitive-online* and the *LP-offline* will serve as an indicator of how well *competitive-online* performs, compared to any possible online algorithms.

- **SPTP-online**: Another baseline algorithm we use is another online algorithm via the SPTP. The difference with this algorithm and that of ours is that the length function of an edge is set to be a function of the congestion level. The length function we choose is the one used in [28] for the OSPF routing, i.e.,

$$len(e) = \begin{cases} 1 & u(e) \in [0, \frac{1}{3}) \\ 3 & u(e) \in [\frac{1}{3}, \frac{2}{3}) \\ 10 & u(e) \in [\frac{2}{3}, \frac{9}{10}) \\ 70 & u(e) \in [\frac{9}{10}, 1) \\ 500 & u(e) \in [1, \frac{11}{10}) \\ 5000 & u(e) \in [\frac{11}{10}, +\infty) \end{cases}$$

  Upon the arrival of each service chain request, we compute the length function based on the current load of each edge, and we route along the Shortest Path Tour accordingly.

- **fewest-hop**: Same as the *SPTP-online* except for the length function. For all edges, we assign its length as 1. By computing the Shortest Path Tour under this value, we obtain a service chain routing which solely minimizes the number of hops from the source to the destination. This baseline algorithm serves as a lower-bound in terms of the number of hops.

4.4.2 Simulation Setup

We set up a simulation to evaluate the effectiveness of our proposed algorithm. For the topology of the physical network, we generate a topology follows the Waxman model, which captures the topology of the intra-domain networks [55]. The topology we use consists of 50 nodes, the capacity of each edge follows a uniform distribution in the range

of $[50, 100]$. We define a total number of six different types of services supported by the physical networks. For each service, we randomly select 10 physical network nodes, representing the locations where the NFs are instantiated.

We use randomly generated service requests. For each service chain request i, we randomly select $1 \leq k_i \leq K$ number of virtual requests, and those NFs are concatenated in a random order. The traffic demand between each network function follows a uniform distribution of $[1, 5]$.

The metrics we evaluate include: 1) the maximum link utilization after we route the service chain requests; 2) the average number of hops for the service chain to reach the destination. For each simulation scenario, we run the simulation for 30 times to obtain the maximal congestion on average and the corresponding 95-percent confidence interval. Also, we measure the average number of hops from the source to the destination. Because the *LP-offline* and the *shortest-walk* are two baselines for the maximum utilization and average number of hops, we only evaluate their performance on the corresponding metrics.

Fig. 4.3 plots the maximal congestion versus the total number of requests being routed on the substrate network, with the maximal length of the chain to be $K = 4$. Compared to the *LP-offline*, the lower-bound on the maximal congestion, the relative gap between the two is in the range of $(9.4\%, 23.3\%)$. Notice that while the *LP-offline* is a lower-bound to the service chain routing problem, the bound is not tight. The reason is that the *LP-offline* will return a fractional routing for the service chain, meaning the congestion will be even lower than the optimal value in the hindsight. This implies that the gap between the *competitive-online* algorithm and the optimal solution in hindsight is potentially smaller than the one presented in Fig. 4.3. Compare to the *SPTP-online* algorithm, the *competitive-offline* delivers a better performance. It minimizes the network congestion a ratio from 0% to 30%.

Fig. 4.4 plots the average number of hops from the source to the destination. In all cases, the proposed algorithm, *competitive-online*, routes the service chain with less than 1.6 additional hops on average when compares to the lower-bound, i.e., *shortest-walk*. While compared to the *SPTP-online*, the number of hops are affected by the amount of service requests routed, or in another word, the load on the substrate network. When the substrate network is under-utilized, the *competitive-online* routes the service chain with as much as 1.16 (or 20.7% in relative) extra number of hops on average; otherwise, when the load is high, it selects the route with 0.69 (or 9.9%) fewer hops. This follows

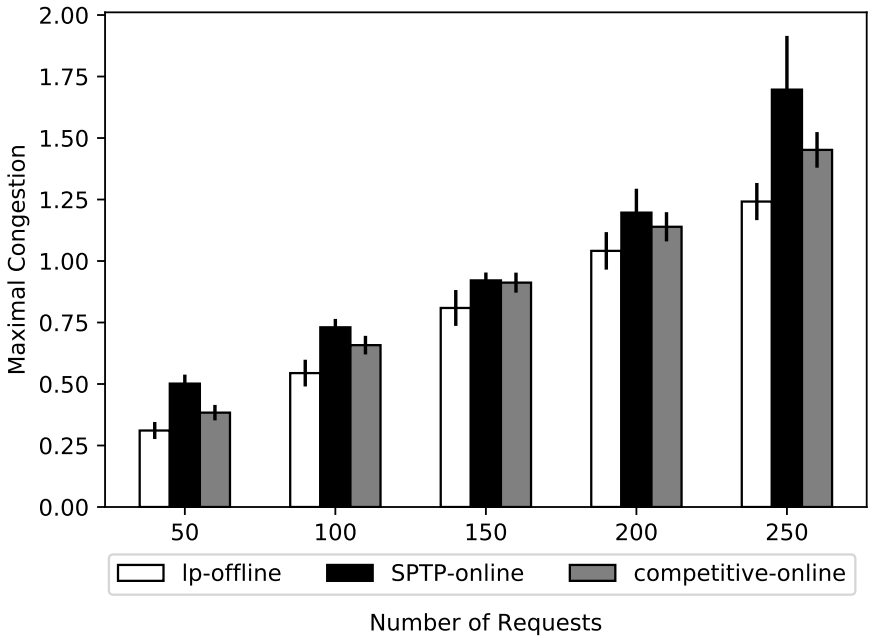

Figure 4.3: Maximal congestion v.s. number of requests

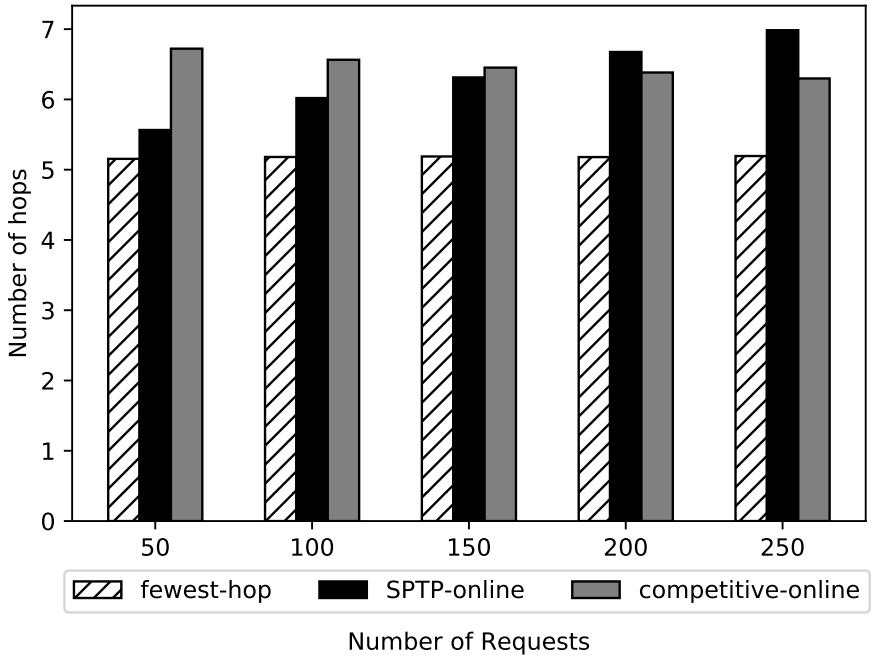

Figure 4.4: Maximal congestion v.s. number of requests

from the fact that the *constant term* in (4.29) depends on ω, the weighted length of the first shortest walk. A larger value for ω would further encourage the routing algorithm to route the service chain along a shorter walk. With a higher amount of traffic in the network, the load on each edge would see an increase. This results in a larger value for ω, and in turn, a larger constant term, and eventually leads the service chain to a walk with a fewer number of hops.

Next, we look at how the maximal length of the service chain would affect the performance. From the previous simulation results, we see that the traffic has an impact on the performance of our algorithm when compared to the baseline algorithm. So, we evaluate the performance of our algorithms under two different conditions: 1) a relatively low load, with $N = 100$ requests and 2) a relatively high load, with $N = 200$ requests.

Fig. 4.5 and Fig. 4.7 plot the congestion versus different maximal service chain length, with a total number of $N = 100$ and $N = 200$ requests respectively. In terms of the maximal congestion, the ratio between *LP-offline* and the *competitive-online* is in a range of 1.17 to 1.27 when $N = 100$, and 1.06 to 1.21 when $N = 200$. Compare to *SPTP-online*, we can see that the *competitive-online* minimizes the congestion by 9% to 11% when $N = 100$ and by 3% to 15% with $N = 200$.

Fig. 4.6 and Fig. 4.8 plots the mean number of hops with respect to the maximal service chain length. We can observe that the load has the same impact on the number of hops on the *competitive-online* algorithm as we see in Fig 4.4: it routes the service chain along the route with more number of hops when the load is lower. Compared to the lower bound, *fewest-hops*, our proposed algorithm needs 1.1 to 1.7 extra hops (or 23% to 29% more hops) to route the service chain, when $N = 100$. Likewise, it needs 0.82 to 1.49 extra hops (or 18% to 25% more hops) when there are 200 hundred requests.

Compared to the *SPTP-online*, we see that our algorithm needs up to 0.57 additional hops (or 10% in comparison) to with $N = 100$. With 200 service requests, *competitive-online* generally results in a fewer of hops. The number of hops levels up with *SPTP-online* when $K = 2$, while it can route with 0.7 fewer hops when $K = 6$.

From this set of simulations, we can conclude that our proposed algorithm is near-optimal in terms of minimizing the congestion, demonstrated by the fact the *competitive-online* algorithm results in 27% higher congestion against an offline lower-bound, while it consistently delivers a better congestion than the *ospf-online* algorithm. As for the minimizing the number of hops needs, the proposed algorithm generally incurs a small

increase in the number of hops to route the service chain. Indeed, compared to the *few-hops*, we can see that our proposed algorithm needs up to less than 1.3 hops on average.

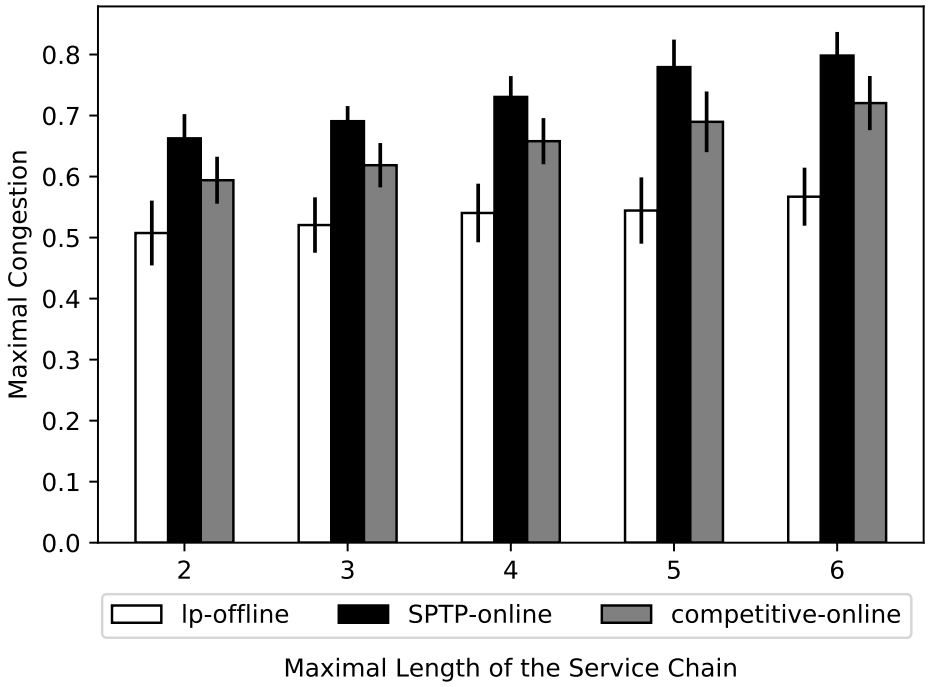

Figure 4.5: Maximal congestion v.s. number of requests

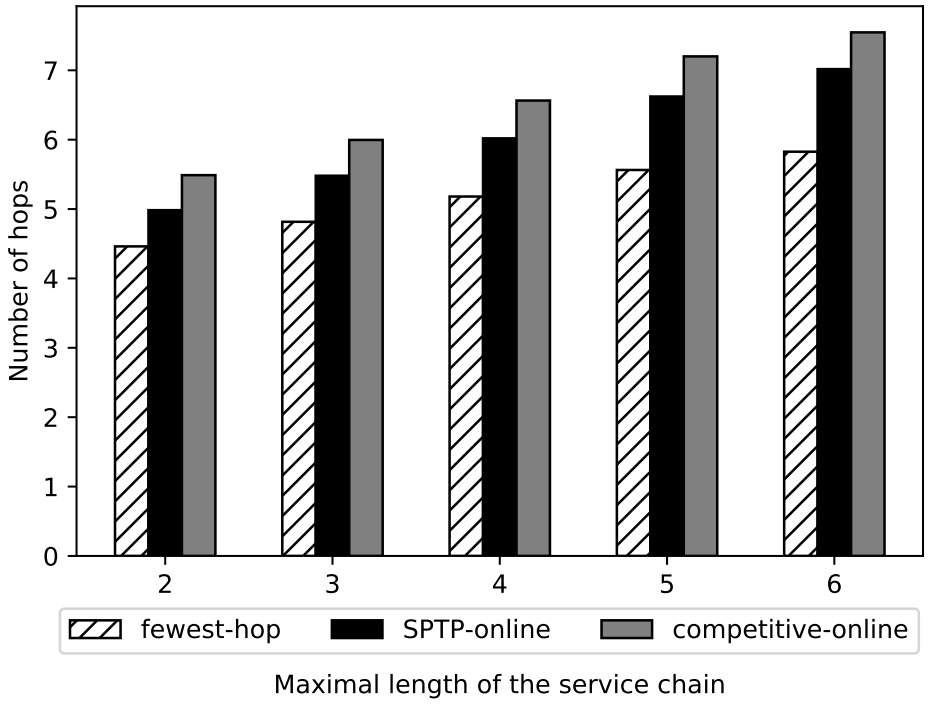

Figure 4.6: Maximal congestion v.s. number of requests

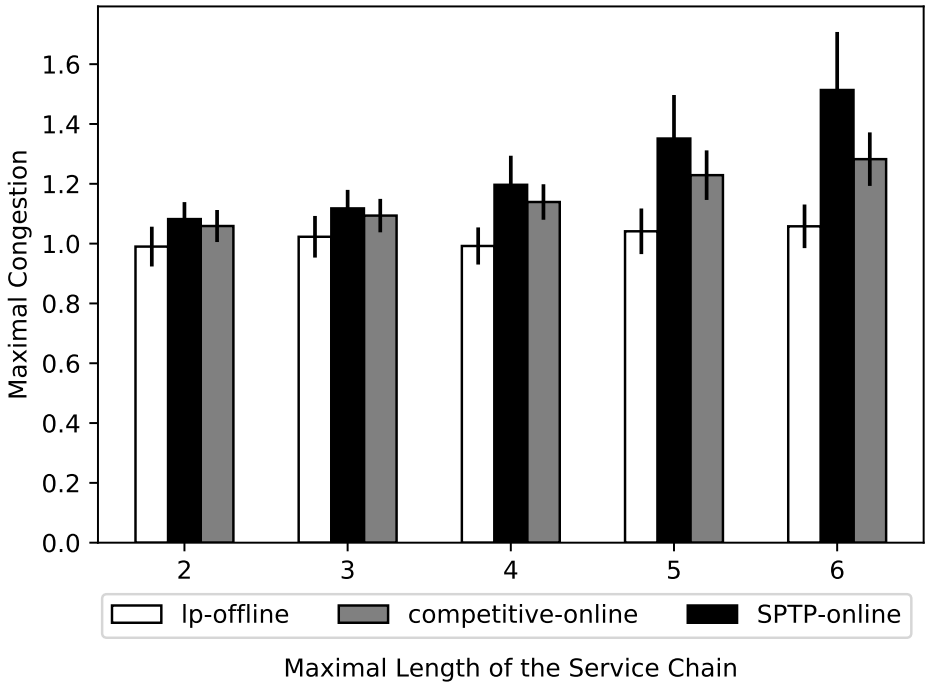

Figure 4.7: Maximal congestion v.s. number of requests

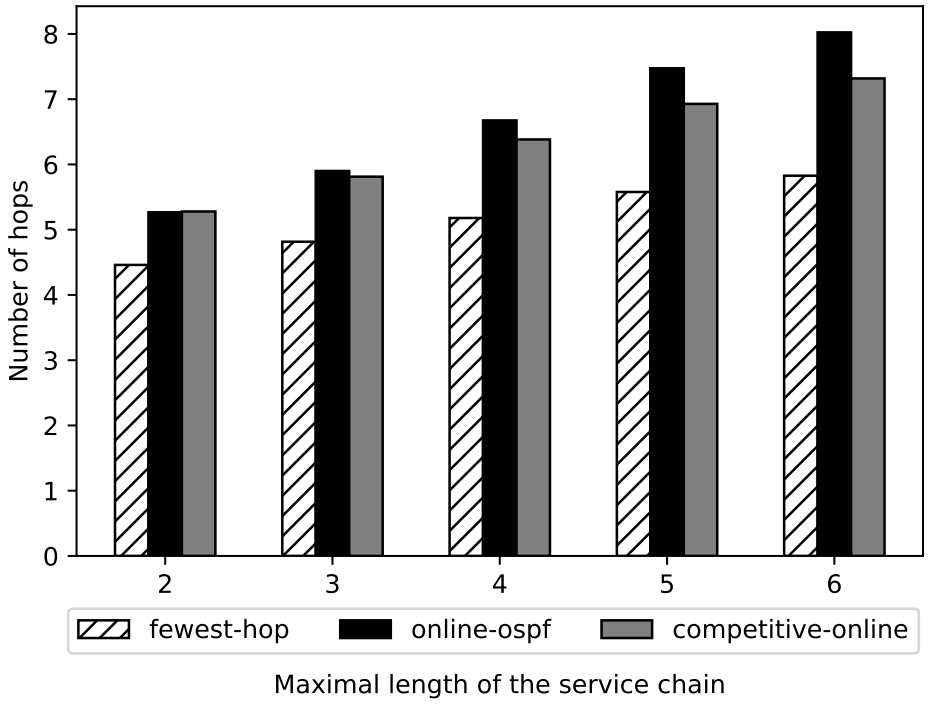

Figure 4.8: Maximal congestion v.s. number of requests

Chapter 5

Conclusions and Future Works

In our work, we have discussed the resource allocation problems in the virtual network environment. A summary of our work is as follows:

First, we presented a spectral clustering based algorithm to partition the virtual network requests into clusters. By transforming a virtual network request into the eigenspace, the traffic intensity could be captured by the Euclidean distance. A clustering on eigenspace could lead to a partitioning of the virtual network request, which would minimize the inter-cluster traffic.

Second, we presented a reconfiguration strategy for virtual network requests. For the virtual network reconfiguration, we jointly consider the workload for substrate nodes and links, as well as migration costs. To avoid the computational complexity to solve a linear programming problem, we propose an FTPAS algorithm base on multi-commodity flow problem.

Third, we developed an online algorithm for the service chain routing in an NFV environment. We presented an efficient algorithm that aims at jointly minimizing network congestion and the number of hops. We prove that this algorithm is $O(logm)$-competitive.

The effectiveness of those strategies is evaluated by the simulations. We believe these methods can help the network operator to make a more informed decision towards resources management in a Virtual Network Environment.

5.1 Future Work

We proposed a new perspective to solve the resource allocation problems in a virtual network environment. Some of the future directions are summarized as follows.

1. **Online Admission Control Policies:** In a virtual network environment, the problem we deal with, in large part, is to embed the virtual network requests onto a physical network with finite resources. As the virtual network request arrives dynamically, often, it is the case that we can not admit all the requests over the physical network. One future research direction is to couple the requests embedding with the admission control policies, in a way that we can admit as many requests as possible, without significant performance degradation. In this context, future research effort could direct toward developing efficient online algorithms with a provably good performance guarantee.

2. **Time Varying Resource Demand:** The work of this thesis assumes the resource demand remains static, and is known in advance. We may further study how to handle the case where the resource demand will change over time, while we do not have complete knowledge of the demand. As the embedding problem can be formulated as the optimization problem, one possible research direction is to incorporate the time-varying factor into the optimization problem and devise efficient algorithm to solve those problems. Another possible research direction is to devise a prediction based algorithm to address the uncertainties on the resource demand.

3. **Virtual Request Condition:** As discussed in Chapter 3, we can achieve a better network performance by partially remapping the virtual nodes and links. In the same spirit, we envision that rerouting can improve the performance of the network function virtualization. However, each reconfiguration comes at a cost: virtual node migration leads to an increasing amount of traffic, while the rerouting dictates setting up new paths and tearing down the existing ones. Thus, one opening question is when to trigger the reconfiguration events.In addition to the periodical reconfiguration or invoke the reconfiguration strategies when the network condition deteriorates to certain conditions, it might be worth exploring new policies to trigger the reconfiguration that can deliver better performance while minimizing the total number of reconfiguration.

BIBLIOGRAPHY

[1] GT-ITM. `http://aiweb.techfak.uni-bielefeld.de/content/bworld-robot-control-software/`.

[2] Thomas Anderson, Larry Peterson, Scott Shenker, and Jonathan Turner. Overcoming the internet impasse through virtualization. *Computer*, 38(4):34–41, 2005.

[3] Konstantin Andreev and Harald Racke. Balanced graph partitioning. *Theory of Computing Systems*, 39(6):929–939, 2006.

[4] James Aspnes, Yossi Azar, Amos Fiat, Serge Plotkin, and Orli Waarts. On-line load balancing with applications to machine scheduling and virtual circuit routing. In *Proceedings of the twenty-fifth annual ACM symposium on Theory of computing*, pages 623–631. ACM, 1993.

[5] Awerbuch et al. The price of routing unsplittable flow. In *Proceedings of STOC*. ACM, 2005.

[6] Albert-László Barabási and Réka Albert. Emergence of scaling in random networks. *science*, 286(5439):509–512, 1999.

[7] Md Faizul Bari, Raouf Boutaba, Rafael Esteves, Lisandro Zambenedetti Granville, Maxim Podlesny, Md Golam Rabbani, Qi Zhang, and Mohamed Faten Zhani. Data center network virtualization: A survey. *IEEE Communications Surveys & Tutorials*, 15(2):909–928, 2013.

[8] Md Faizul Bari, Shihabur Rahman Chowdhury, Reaz Ahmed, and Raouf Boutaba. On orchestrating virtual network functions. In *Network and Service Management (CNSM), 2015 11th International Conference on*, pages 50–56. IEEE, 2015.

[9] Michael Till Beck and Juan Felipe Botero. Coordinated allocation of service function chains. In *Global Communications Conference (GLOBECOM), 2015 IEEE*, pages 1–6. IEEE, 2015.

[10] Shireesh Bhat et al. Service-concatenation routing with applications to network functions virtualization. In *Proceedings of ICCCN*. IEEE, 2017.

[11] Paul S Bradley, Olvi L Mangasarian, and W Nick Street. Clustering via concave minimization. In *Advances in neural information processing systems*, pages 368–374, 1997.

[12] PS Bradley, KP Bennett, and Ayhan Demiriz. Constrained k-means clustering. *Microsoft Research, Redmond*, pages 1–8, 2000.

[13] Aydın Buluç, Henning Meyerhenke, Ilya Safro, Peter Sanders, and Christian Schulz. Recent advances in graph partitioning. In *Algorithm Engineering*, pages 117–158. Springer, 2016.

[14] Zhiping Cai, Fang Liu, Nong Xiao, Qiang Liu, and Zhiying Wang. Virtual network embedding for evolving networks. In *Global Telecommunications Conference (GLOBECOM 2010), 2010 IEEE*, pages 1–5. IEEE, 2010.

[15] Xiang Cheng, Sen Su, Zhongbao Zhang, Hanchi Wang, Fangchun Yang, Yan Luo, and Jie Wang. Virtual network embedding through topology-aware node ranking. *ACM SIGCOMM Computer Communication Review*, 41(2):38–47, 2011.

[16] NM Mosharaf Kabir Chowdhury and Raouf Boutaba. Network virtualization: state of the art and research challenges. *IEEE Communications magazine*, 47(7), 2009.

[17] NM Mosharaf Kabir Chowdhury and Raouf Boutaba. A survey of network virtualization. *Computer Networks*, 54(5):862–876, 2010.

[18] NM Mosharaf Kabir Chowdhury, Muntasir Raihan Rahman, and Raouf Boutaba. Virtual network embedding with coordinated node and link mapping. In *INFOCOM 2009, IEEE*, pages 783–791. IEEE, 2009.

[19] Boutheina Dab, Ilhem Fajjari, Nadjib Aitsaadi, and Guy Pujolle. Vnr-ga: Elastic virtual network reconfiguration algorithm based on genetic metaheuristic. In *Global Communications Conference (GLOBECOM), 2013 IEEE*, pages 2300–2306. IEEE, 2013.

[20] Patricia Takako Endo, Andre Vitor de Almeida Palhares, Nadilma Nunes Pereira, Glauco Estacio Goncalves, Djamel Sadok, Judith Kelner, Bob Melander, and Jan-Erik Mangs. Resource allocation for distributed cloud: concepts and research challenges. *IEEE network*, 25(4), 2011.

[21] Guy Even, Matthias Rost, and Stefan Schmid. An approximation algorithm for path computation and function placement in SDNs. In *International Colloquium on Structural Information and Communication Complexity*, pages 374–390. Springer, 2016.

[22] Ilhem Fajjari, Nadjib Aitsaadi, Guy Pujolle, and Hubert Zimmermann. Vnr algorithm: A greedy approach for virtual networks reconfigurations. In *Global Telecommunications Conference (GLOBECOM 2011), 2011 IEEE*, pages 1–6. IEEE, 2011.

[23] Michalis Faloutsos, Petros Faloutsos, and Christos Faloutsos. On power-law relationships of the internet topology. In *ACM SIGCOMM computer communication review*, volume 29, pages 251–262. ACM, 1999.

[24] Xincai Fei, Fangming Liu, Hong Xu, and Hai Jin. Adaptive vnf scaling and flow routing with proactive demand prediction. In *INFOCOM 2018-The 37th Annual IEEE International Conference on Computer Communications*, pages 1–9, 2018.

[25] Festa et al. Solving the shortest path tour problem. *European Journal of Operational Research*, 2013.

[26] Paola Festa. The shortest path tour problem: problem definition, modeling, and optimization. In *Proceedings of INOC*, 2009.

[27] Andreas Fischer, Juan Felipe Botero, Michael Till Beck, Hermann De Meer, and Xavier Hesselbach. Virtual network embedding: A survey. *IEEE Communications Surveys & Tutorials*, 15(4):1888–1906, 2013.

[28] Bernard Fortz and Mikkel Thorup. Optimizing ospf/IS-IS weights in a changing world. *IEEE journal on selected areas in communications*, 20(4):756–767, 2002.

[29] Long Gong, Yonggang Wen, Zuqing Zhu, and Tony Lee. Toward profit-seeking virtual network embedding algorithm via global resource capacity. In *INFOCOM, 2014 Proceedings IEEE*, pages 1–9. IEEE, 2014.

[30] Bo Han et al. Network function virtualization: Challenges and opportunities for innovations. *IEEE Communications Magazine*, 2015.

[31] Bruce Hendrickson and Robert Leland. An improved spectral graph partitioning algorithm for mapping parallel computations. *SIAM Journal on Scientific Computing*, 16(2):452–469, 1995.

[32] Juliver Gil Herrera and Juan Felipe Botero. Resource allocation in NFV: A comprehensive survey. *IEEE Transactions on Network and Service Management*, 13(3):518–532, 2016.

[33] Ines Houidi, Wajdi Louati, Walid Ben Ameur, and Djamal Zeghlache. Virtual network provisioning across multiple substrate networks. *Computer Networks*, 55(4):1011–1023, 2011.

[34] Liang Huang, Kwangho Park, and Ying-Cheng Lai. Information propagation on modular networks. *Physical Review E*, 73(3):035103, 2006.

[35] Kaustubh Joshi et al. Network function virtualization. *IEEE Internet Computing*, 2016.

[36] George Karakostas. Faster approximation schemes for fractional multicommodity flow problems. *ACM Transactions on Algorithms (TALG)*, 4(1):13, 2008.

[37] George Karypis and Vipin Kumar. Metis–unstructured graph partitioning and sparse matrix ordering system, version 2.0. 1995.

[38] George Karypis and Vipin Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on scientific Computing*, 20(1):359–392, 1998.

[39] Brian W Kernighan and Shen Lin. An efficient heuristic procedure for partitioning graphs. *The Bell system technical journal*, 49(2):291–307, 1970.

[40] Jian-Jhih Kuo et al. Service chain embedding with maximum flow in software defined network and application to the next-generation cellular network architecture. In *Proceedings of INFOCOM*. IEEE, 2017.

[41] Aris Leivadeas, Chrysa Papagianni, and Symeon Papavassiliou. Efficient resource mapping framework over networked clouds via iterated local search-based request partitioning. *IEEE Transactions on Parallel and Distributed Systems*, 24(6):1077–1086, 2013.

[42] Tachun Lin, Zhili Zhou, Massimo Tornatore, and Biswanath Mukherjee. Demand-aware network function placement. *Journal of Lightwave Technology*, 34(11):2590–2600, 2016.

[43] Tamás Lukovszki and Stefan Schmid. Online admission control and embedding of service chains. In *International Colloquium on Structural Information and Communication Complexity*, pages 104–118. Springer, 2015.

[44] Wenrui Ma, Oscar Sandoval, Jonathan Beltran, Deng Pan, and Niki Pissinou. Traffic aware placement of interdependent NFV middleboxes. In *INFOCOM 2017-IEEE Conference on Computer Communications, IEEE*, pages 1–9. IEEE, 2017.

[45] James MacQueen et al. Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, volume 1, pages 281–297. Oakland, CA, USA, 1967.

[46] Sevil Mehraghdam, Matthias Keller, and Holger Karl. Specifying and placing chains of virtual network functions. In *Cloud Networking (CloudNet), 2014 IEEE 3rd International Conference on*, pages 7–13. IEEE, 2014.

[47] Xiaoqiao Meng, Vasileios Pappas, and Li Zhang. Improving the scalability of data center networks with traffic-aware virtual machine placement. In *INFOCOM, 2010 Proceedings IEEE*, pages 1–9. IEEE, 2010.

[48] Maurizio Naldi. Connectivity of waxman topology models. *Computer communications*, 29(1):24–31, 2005.

[49] Andrew Y Ng, Michael I Jordan, and Yair Weiss. On spectral clustering: Analysis and an algorithm. In *Advances in neural information processing systems*, pages 849–856, 2002.

[50] Muntasir Raihan Rahman, Issam Aib, and Raouf Boutaba. Survivable virtual network embedding. Springer.

[51] Sen Su, Zhongbao Zhang, Xiang Cheng, Yiwen Wang, Yan Luo, and Jie Wang. Energy-aware virtual network embedding through consolidation. In *Computer Communications Workshops (INFOCOM WKSHPS), 2012 IEEE Conference on*, pages 127–132. IEEE, 2012.

[52] Phuong Nga Tran and Andreas Timm-Giel. Reconfiguration of virtual network mapping considering service disruption. In *Communications (ICC), 2013 IEEE International Conference on*, pages 3487–3492. IEEE, 2013.

[53] Tri Trinh, Hiroshi Esaki, and Chaodit Aswakul. Quality of service using careful overbooking for optimal virtual network resource allocation. In *Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTI-CON), 2011 8th International Conference on*, pages 296–299. IEEE, 2011.

[54] Ulrike Von Luxburg. A tutorial on spectral clustering. *Statistics and computing*, 17(4):395–416, 2007.

[55] Bernard M Waxman. Routing of multipoint connections. *IEEE journal on selected areas in communications*, 6(9):1617–1622, 1988.

[56] Peng Wu, Yong Cui, Jianping Wu, Jiangchuan Liu, and Chris Metz. Transition from ipv4 to ipv6: A state-of-the-art survey. *IEEE Communications Surveys & Tutorials*, 15(3):1407–1424, 2013.

[57] Ming Xia, Meral Shirazipour, Ying Zhang, Howard Green, and Attila Takacs. Network function placement for NFV chaining in packet/optical datacenters. *Journal of Lightwave Technology*, 33(8):1565–1570, 2015.

[58] Yufeng Xin, Ilia Baldine, Anirban Mandal, Chris Heermann, Jeff Chase, and Aydan Yumerefendi. Embedding virtual topologies in networked clouds. In *Proceedings of the 6th international conference on future internet technologies*, pages 26–29. ACM, 2011.

[59] Zichuan Xu, Weifa Liang, Alex Galis, Yu Ma, Qiufen Xia, and Wenzheng Xu. Throughput optimization for admitting NFV-enabled requests in cloud networks. *Computer Networks*, 2018.

[60] Zichuan Xu, Weifa Liang, Meitian Huang, Mike Jia, Song Guo, and Alex Galis. Approximation and online algorithms for NFV-enabled multicasting in SDNs. In *Distributed Computing Systems (ICDCS), 2017 IEEE 37th International Conference on*, pages 625–634. IEEE, 2017.

[61] Minlan Yu, Yung Yi, Jennifer Rexford, and Mung Chiang. Rethinking virtual network embedding: substrate support for path splitting and migration. *ACM SIGCOMM Computer Communication Review*, 38(2):17–29, 2008.

[62] Yong Zhu and Mostafa H Ammar. Algorithms for assigning substrate network resources to virtual network components. In *INFOCOM*, volume 1200, pages 1–12, 2006.