Resource Orchestration in Softwarized Networks

by

Md. Faizul Bari

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Computer Science

Waterloo, Ontario, Canada, 2018

Examining Committee Membership

The following served on the Examining Committee for this thesis. The decision of the Examining Committee is by majority vote.

External Examiner	Yashar Ganjali Associate Professor Department of Computer Science University of Toronto
Supervisor	Raouf Boutaba Professor David R. Cheriton School of Computer Science University of Waterloo
Internal Member	Martin Karsten Associate Professor David R. Cheriton School of Computer Science University of Waterloo
Internal Member	Bernard Wong Associate Professor David R. Cheriton School of Computer Science University of Waterloo
Internal-external Member	Pin-Han Ho Professor Department of Electrical & Computer Engineering University of Waterloo

Author's Declaration for Electronic Submission of Thesis

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Md. Faizul Bari

Abstract

Network softwarization is an emerging research area that is envisioned to revolutionize the way network infrastructure is designed, operated, and managed today. Contemporary telecommunication networks are going through a major transformation, and softwarization is recognized as a crucial enabler of this transformation by both academia and industry. Softwarization promises to overcome the current ossified state of Internet network architecture and evolve towards a more open, agile, flexible, and programmable networking paradigm that will reduce both capital and operational expenditures, cut-down time-to-market of new services, and create new revenue streams. Software-Defined Networking (SDN) and Network Function Virtualization (NFV) are two complementary networking technologies that have established themselves as the cornerstones of network softwarization. SDN decouples the control and data planes to provide enhanced programmability and faster innovation of networking technologies. It facilitates simplified network control, scalability, availability, flexibility, security, cost-reduction, autonomic management, and fine-grained control of network traffic. NFV utilizes virtualization technology to reduce dependency on underlying hardware by moving packet processing activities from proprietary hardware middleboxes to virtualized entities that can run on commodity hardware. Together SDN and NFV simplify network infrastructure by utilizing standardized and commodity hardware for both compute and networking; bringing the benefits of agility, economies of scale, and flexibility of data centers to networks.

Network softwarization provides the tools required to re-architect the current network infrastructure of the Internet. However, the effective application of these tools requires efficient utilization of networking resources in the softwarized environment. Innovative techniques and mechanisms are required for all aspects of network management and control. The overarching goal of this thesis is to address several key resource orchestration challenges in softwarized networks. The resource allocation and orchestration techniques presented in this thesis utilize the functionality provided by softwarization to reduce operational cost, improve resource utilization, ensure scalability, dynamically scale resource pools according to demand, and optimize energy utilization.

Acknowledgements

First and foremost, I thank Allah, the Exalted in Might, the Wisest, and the Omniscient, for bringing me into existence and enabling me to complete this work.

I express my warmest gratitude to my supervisor, Professor Raouf Boutaba, for supporting me with a continuous stream of intellectual, moral, and financial assistance. He has always encouraged me to pursue my ideas and provided invaluable feedback to improve the quality of my research. I am especially thankful for his honest feedback that had developed me as a researcher and a person. He is not only a true mentor to me in the avenues of research but also an invaluable friend in the ways of life. Finally, I want to extend my gratitude and acknowledgments to Professor Yashar Ganjali, Professor Martin Karsten, Professor Bernard Wong, and Professor Pin-Han Ho for their insightful comments and constructive criticisms on this work.

I am truly grateful to my father, Dr. Ali Hossen Bhuyian, and my mother, Dr. Monowara Begum, for constantly inspiring me in the quest for knowledge, without which it would be impossible for me to complete this work. Their unconditional love and support enabled me to secure my career plans. I owe my deepest gratitude to my colleagues Shihab, Arup, Rabbani, Faten, and Qi for their support, enlightening discussions, and a lot of cheerful moments. I thank all my teachers for their inspiration, wisdom and endless efforts. Last but not least, I thank everyone who has contributed to this thesis, directly or indirectly.

Dedication

Dedicated to my father Dr. Ali Hossen Bhuyian.

Table of Contents

List of Tables

List of Figures

List of Algorithms

Chapter 1

Introduction

The proliferation of the Internet has created a digital society where online communication has become an essential and ubiquitous part of our lives. Online communication services have evolved into an infrastructure utility that enables everyone and everything to communicate with each other. The Internet's history is marked by continuous evolution and growth of user demand and capacity; coupled with the explosive growth of new services and applications provided on top of it. A direct consequence of this phenomenon is observed through the tremendous growth experienced by the Internet over the last 30 years [1]. In 2016, per year global Internet traffic was at 1.2 ZB, and is predicted to reach 3.3 ZB per year by 2021; indicating a threefold increase within a duration of 5 years [2]. This growth is driven by the rapid surge of devices like smartphones, smart-things, tablets, and laptops connecting to the Internet and emerging technologies like cloud computing, storage synchronization services, photo-intensive social networks, big data, on-demand video streaming, online gaming, Internet of Things (IoT), and Virtual/Augmented Reality (VR/AR) applications. These new applications and technologies require ultra-low latency and significantly higher bandwidth; imposing new challenges like on-demand scaling, high-resiliency, and on-the-fly service configuration for the contemporary and future network architectures.

Networking technologies have always faced requirements from multiple stakeholders like network operators, service providers, content providers, application developers, and end-users. Most often these requirements do not comply with each other, *e.g.*, an end-user may want to have the best possible Quality of Service (QoS), while the network operator prefers to minimize cost for providing just enough QoS. Contemporary and emerging Internet applications and services have highly diverse requirements; for example, streaming one minute of High Definition (HD) video requires sending around 4 megabytes of data; in

contrast, streaming one minute of VR video takes hundreds of megabytes [3]. With the increasing popularity of VR, a much higher bandwidth requirement will be imposed on the network infrastructure. In addition, IoT is encouraging the rapid deployment of Internet-connected devices that are often controlled and monitored through the Internet. Most IoT applications require ultra-low latency and at the same time generate mountains of data that must be transported through the network for cloud-based storage and analytics. The number of IoT devices is estimated to reach 20–50 billion by 2020 [4]; therefore, the demand for both ultra-low latency and high bandwidth network will continue to rise. The requirements mentioned above demand sophisticated technologies to diversify and enhance network transport's flexibility, agility, and capacity.

Technological innovations in the Internet's architecture have not progressed at the same pace as that of the services and applications provided on top of it. The underlying infrastructure and technology have mostly remained ossified at the same stage for decades. The primary reason behind this ossification is the unnecessary coupling between the service-oriented features and transport-oriented infrastructure of the Internet. A network operator is constrained within the functionalities provided by equipments of the underlying infrastructure. Whenever a new network requirement is identified, a Request For Proposal (RFP) is issued by the network operator. Several vendors can offer solutions conforming to the requirements, and then the network operator picks one of them. Vendors typically use specialized hardware and closed-source software to retain their market position and optimize their product for both cost and performance. Most often network operators do not have enough flexibility or programmability to add, modify, or remove any built-in functionality of the appliance. This approach introduces a dependency on proprietary, closed, and vertically-integrated networking hardware and software; creates vendor and platform lock-in, and restrains the operator within limited options to upgrade an existing appliance. In most cases, network operators must purchase a new product, possibly from the same vendor, to introduce new services in their networks.

Apart from the issues mentioned above, another category of issues arises due to the way network capacity is planned and the rampant rise of Internet traffic. The traditional approach to network capacity planning involves over-provisioning of resources to meet estimated future requirements. The current rapid growth of traffic forces network planners to pre-allocate significant amount of additional resources that reduces the overall utilization of the network. To overcome this challenge, the current network architecture must circumvent the hardware-centric design paradigm and transition towards an open-source and programmable software-centric network paradigm, where network resources are provisioned autonomously and dynamically based on actual traffic volume. For example, Google developed a private WAN, called B4 [5], to connect its data-centers around the globe us-

ing commodity switches built from merchant silicon and software-based control platform. B4 provided substantial cost-saving by enabling Google to run many network links near 100% utilization for extended periods of time. A move from closed-source and proprietary hardware-based approach to open-source and programmable software-enabled networking will enable network operators to adapt their network in the most suitable and optimal manner to reduce cost and provide better QoS.

The lack of programmability and closed-source nature of network appliances lead to complex and inflexible network control and management systems, cripple service flexibility and agility, prolong time to market for new services, and hinder innovation and adoption of new technologies. Contemporary and emerging Internet applications and services require a paradigm shift in the networking architecture to ensure a fast, secure, reliable, flexible, and agile underlying network infrastructure. Network softwarization [6] is an emerging area of research that can address the limitations mentioned above. It offers a new networking paradigm that has the potential to overcome inflexibilities in current network architecture, redesign and significantly enhance service provisioning, reduce capital (CAPEX) and operational (OPEX) expenditures through better resource utilization, provide self-management, and bring cloud-like agility, flexibility and economies of scale to the network infrastructure. Network softwarization revolutionizes the way network infrastructure is designed, operated, and managed by facilitating the unification of network and cloud applications. This revolution is further driven by the wide availability and adoption of open-source software platforms like OpenStack [7], OpenDaylight [8] and OPNFV [9]. Network softwarization enables a network operator to reduce CAPEX by using commodity servers and switches instead of vendor-specific proprietary hardware. Similar to cloud computing, softwarization empowers an network operator to open up the infrastructure with programmable interfaces for third-party content providers and software developers as a way to generate new revenue streams. To fully exploit the benefits of network softwarization, innovative techniques and mechanisms are required for all aspects of network management and control.

The overarching theme of this thesis is the development of resource allocation and orchestration techniques for addressing several critical obstacles in softwarized networks. In the rest of this chapter, an overview of key enabling technologies for softwarization are provided; then several key challenges in softwarized networks are presented. Next, the main contributions of this dissertation are summarized, and finally, an outline of the next chapters is provided.

1.1 Enabling Technologies for Softwarization

Transitioning from a hardware-centric to a software-centric paradigm requires the disaggregation of packet processing logic from the underlying hardware switching fabric. The infrastructure should utilize standardized and commodity hardware whenever possible; this will eliminate vendor lock-ins, and new technological innovations will be easier to integrate. Moreover, the software for processing network packets can evolve independently of the underlying hardware equipment. A software-driven infrastructure enables a higher degree of operational automation through standardized APIs, and SDKs. Different software-based network components can share the commodity hardware pool; thereby reducing cost and increasing utilization. In addition, network capacity can be planned and scaled in a fluid manner by adding additional hardware resources only when they are required. The transformation towards a software-centric network architecture or network softwarization is envisioned to drive the next evolution of Internet's underlying network architecture. This section provides a brief description of Software-Defined Networking (SDN) [10] and Network Function Virtualiztion (NFV) [11], which have emerged as the primary enabling technologies for softwarization.

SDN and NFV are two complementary networking technologies that have established themselves as the cornerstones of network softwarization. SDN proposes to decouple the control and data planes of the network by stripping the control logic from network devices. The control logic is provisioned on a logically centralized entity called the network controller that maintains a global network view and is typically deployed on a single or cluster of servers. In traditional networks, the data and control planes are coupled in the hardware. The control plane runs distributed routing protocols like OSPF, IS-IS, or BGP to compute packet forwarding rules that are utilized by the data plane to forward traffic. In most cases, network devices are proprietary, closed-source, and vertically integrated. It is very complicated or in most cases impossible to deploy a new routing protocol without upgrading networking devices. In an SDN, the data plane is implemented by simple commodity network devices that can efficiently forward traffic based on forwarding table rules. The rules in the forwarding table are computed and installed by the controller using a standardized communication protocol like OpenFlow [12]. SDN promises flexibility and programmability to dynamically re-configure the data plane in real-time. Although software has been deployed for decades to control networks, SDN enables software to define network behaviors rather than merely controlling network operations. It facilitates simplified network control, scalability, availability, flexibility, security, cost-reduction, failure-restoration, autonomic management, and fine-grained control of network traffic.

On the other hand, NFV's objective is to re-design the service provisioning architecture

by virtualizing network functions that were previously provided by dedicated hardware appliances or middleboxes. NFV utilizes virtualization technology to reduce dependency on underlying hardware by moving packet processing activities from proprietary hardware middleboxes to virtualized entities called Virtual Network Functions (VNFs) that can be instantiated on commodity hardware. In traditional networks, services are materialized by utilizing various types of middleboxes which are usually procured from third-party vendors. Middleboxes are typically vendor-specific, vertically-integrated, and closed-source. Network operators are locked-in with a vendor once its middleboxes are integrated into the network. In addition, middleboxes require specially trained personnel for on-going operation and maintenance which leads to high OPEX [13, 14]. Moreover, the closed-source nature of middleboxes hinders innovation and prolongs the time-to-market of new services. NFV resolves these issues by decoupling a middlebox's packet processing logic from the underlying hardware through virtualization. VNFs can be deployed in data centers, telecommunication central offices, network nodes, and even in customer premises equipments. SDN and NFV simplify network infrastructure by utilizing inexpensive commodity hardware for both compute and networking; bringing the benefits of agility, economies of scale, and flexibility of data centers to networks.

All together, network softwarization provides the tools required to re-architect the current network infrastructure of the Internet. However, the effective application of these tools requires efficient utilization of network resources in the softwarized environment. SDN provides control over the paths taken by different traffic flows; NFV enables software-based packet processing to realize network services, and together they provide the ability to dynamically orchestrate resources to satisfy application requirements, achieve energy efficiency, and lower operational costs through optimal resource allocation and orchestration.

1.2 Challenges in Softwarized Networks

This section provides an overview of the key research challenges of resource orchestration in softwarized networks addressed in this thesis. First, we discuss the performance and scalability challenges for multi-controller deployment in SDN. Then we describe several issues related to resource allocation and energy efficiency for network function orchestration in softwarized networks.

1.2.1 Multiple Controllers in SDN

A common misconception about SDN based deployment is that a single physical server operates as the centralized controller [15, 16]. However, the controller is only *logically* centralized; it can be physically distributed. In most cases, a single server will not be enough to meet the processing, delay, or fault-tolerance requirements of a network [17, 18, 19]. First, the number of events like flow-setup requests and packet counter updates grows with the number of switches, flows, and traffic. A single physical server may not have enough processing capacity to handle all the incoming requests. Second, a single controller will always be too far from some switches if the network spans a moderately large geographic area. Hence, multiple controllers will be required to keep the switch-to-controller latency within acceptable limits. Finally, a single controller introduces a single-point-of-failure for the entire network. So, in most SDN deployments, the logically centralized controller is realized by deploying it as a distributed entity. The controller can be distributed using different techniques, for example, a good number of research works like FlowVisor [20], Onix [21], Kandoo [22], SiBF [23], HyperFlow [24], and Devolved-Controllers [25] explored the system level technicalities of this issue and proposed techniques to distribute the controller. In this work, we focus on the development of efficient algorithms to dynamically determine the best locations to place multiple controllers within a network.

In an SDN network, the logically centralized controller maintains a global network view and controls all network devices. Whenever a switch or router receives a new flow, it first checks its local flow-table(s) for a matching forwarding rule. If no matching rule is found, then it requests the controller for a traffic forwarding rule. Based on the properties of the flow, the controller determines the appropriate route for the traffic and installs forwarding rules on all devices on the path. From that point on, all switches on the path handle packets in that flow using the newly installed rules. The flow-setup process is depicted in Figure 1.1, and the time required to perform this operation is commonly known as the flow-setup time. It includes the controller's processing delay to determine the forwarding rules and maximum Round-Trip-Time (RTT) between any switches on the path and the controller. The controller's processing delay depends on the number of pending tasks at that particular time-instance and the computational overhead required to determine the forwarding rules. Several studies have shown that the processing delay at the controller increases exponentially once the number of pending requests exceeds a certain threshold [26, 27, 28]. The RTT component of the flow-setup time depends on the network topology, maximum network distance between any switch on the path and the controller, and the amount of congestion on the paths between the switches and controller.

In large-scale WAN and data center deployments, a single controller will certainly lead

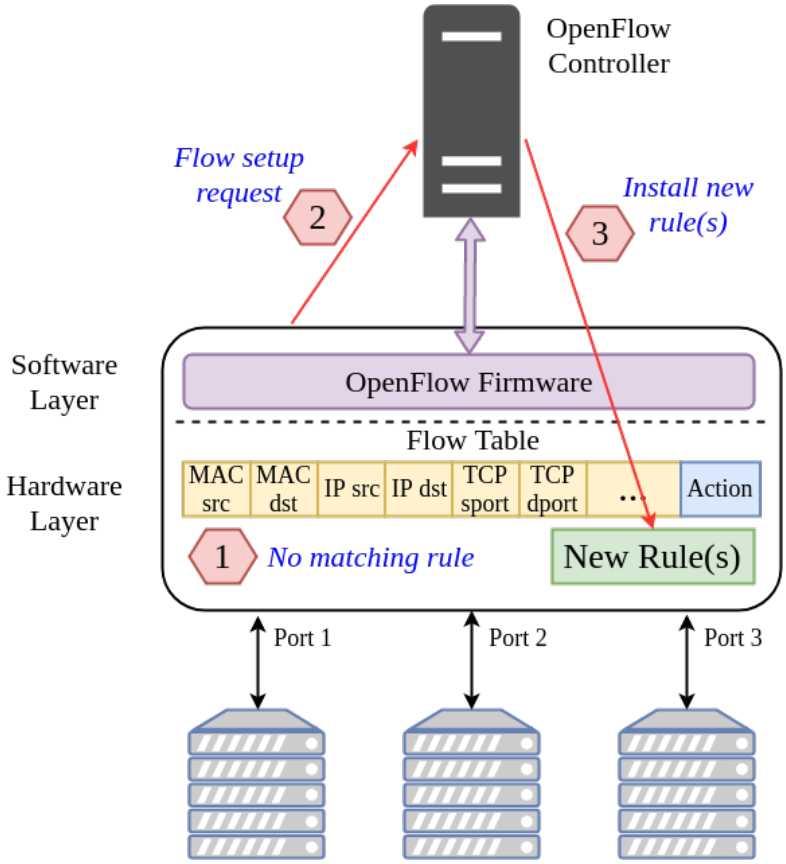


Figure 1.1: Flow setup process

to performance and scalability issues [15]. Deploying multiple controllers is an effective solution to this problem. Figure 1.2 and Figure 1.3 show the same network with a single and then with multiple controllers, respectively. With multiple controllers, the network is effectively divided into zones where each controller takes the responsibility to control the switches in its own zone. However, to efficiently utilize multiple controllers, we need to dynamically adapt the number and locations of controllers according to demand fluctuations and topological characteristics of the network. On one hand, the number of controllers should be sufficient to handle network load, and their locations should ensure acceptable switch-to-controller latencies. On the other hand, the deployment of multiple controllers requires regular state synchronization between the controllers to maintain a consistent global network view [19]. The overhead for state synchronization can be significant if the number of controllers in the network is large. In addition, network dynamicity is a major concern as

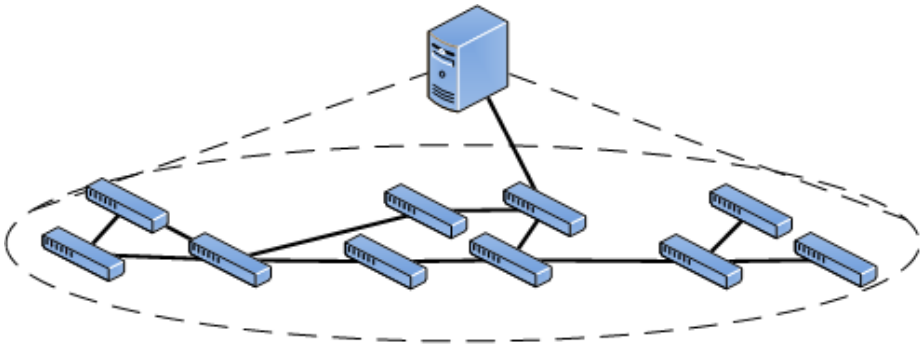


Figure 1.2: Network with a single controller

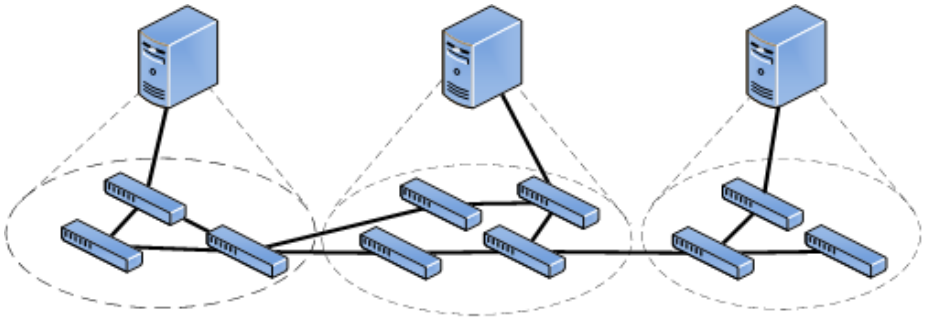


Figure 1.3: Network with multiple controllers

traffic pattern and volume in both WAN and data center environments exhibit significant spatial and temporal fluctuations [29, 30, 31]. The controller provisioning scheme needs to react to network "hotspots" and dynamically re-adjust the number and locations of controllers to ensure an acceptable flow-setup time. A static controller placement provisioned for peak usage across all regions will require a large number of controllers and incur a high state synchronization overhead. Alternatively, if the number of provisioned controllers is not sufficient to handle the network load, then the performance of all applications and services provided by the network will degrade. Hence, we need to find the right trade-off between performance, scalability, and inter-controller state synchronization overhead to dynamically provision multiple controllers in an SDN.

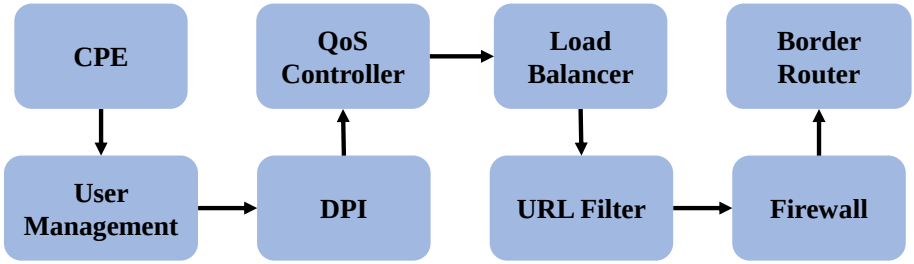
1.2.2 Service Function Chain (SFC) Orchestration

A second challenging issue for softwarized networks is the network service provisioning platform [6, 11]. Contemporary communication networks deploy a diverse set of network services to achieve different types of security and performance objectives [32, 14]. These services are materialized by utilizing various types of hardware appliances or middleboxes [13, 14]. Examples of such middleboxes include firewall, proxy, WAN optimizer, Evolved Packet Core (EPC), IP Multimedia Subsystem (IMS), Intrusion Detection System (IDS) and Intrusion Prevention System (IPS). These middleboxes have become an integral part of modern networks to such an extend that several recent studies found the number of middleboxes to be comparable to the number of routers and switches in enterprise and data center networks [14, 33, 34]. Although, middleboxes are ubiquitously deployed in modern networks, they result in a lot of management and operational complexities along with significant CAPEX and OPEX [35]. The primary reason behind this situation can be attributed to the vendor-specific, vertically-integrated, and closed-source nature of middleboxes.

Hardware middleboxes are expensive and require specially trained personnel for deployment and maintenance. The closed and proprietary nature of traditional middleboxes makes it very challenging and cumbersome for the network operator to introduce new services [11]. It is often impossible to add new functionality to an existing middlebox. In many cases, the network operator is compelled to upgrade or purchase new hardware. Today's networks are facing an ever-increasing requirement to introduce diverse and often short-lived services to retain customer base. These services often require the purchase of new network equipment that is operated only for a short time-span; sometimes the equipment becomes obsolete even before any profit is earned from the offered service [11]. In addition, diverse services demand rapidly changing skill-sets from the operational and management team members of the telecommunication network. Moreover, the telecommunication industry is finding it difficult to raise their profit margin due to ever increasing competition with Over-The-Top (OTT) service providers, as increasing service prices will certainly lead to customer churn [36, 11]. To overcome these issues, a flexible, agile, and innovative service provisioning platform is required that can reduce both CAPEX and OPEX for network services and provide fast and simple mechanisms to introduce new services in the network.

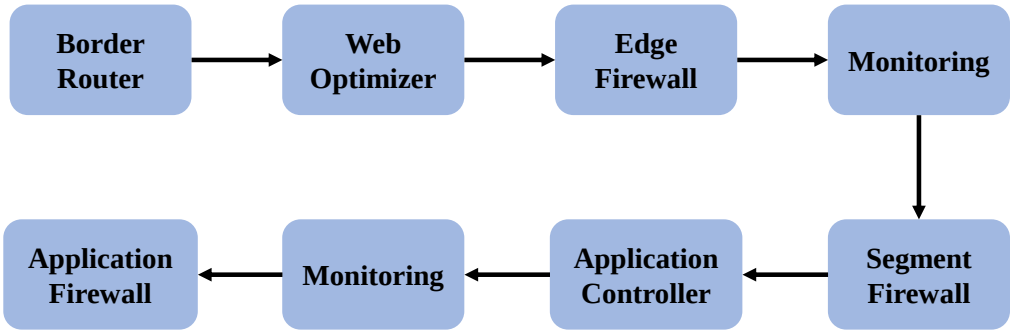
Network services are composed by arranging multiple middleboxes in a specific order to assemble a processing pipeline. Network traffic passes through this pipeline, procuring multiple stages of middlebox processing to ensure a particular security or performance objective [32, 14]. For example, a traffic may need to go through a firewall, then an IDS, and finally through a proxy [37]. This process is very common for middleboxes and is

typically called *Service Function Chaining (SFC)* [38]. SFCs are used to provide various types of user-facing services that are crucial for generating revenue and avoiding customer churn. Different types of SFCs are customarily used for everyday operations in modern networks. The IETF Network and Service Chaining Working Group has several IETF drafts showcasing middlebox chaining use-cases in ISP [39], mobile [40], and data center networks [41]. Figure 1.4 depicts three such SFCs that are frequently deployed in real-world networks: (i) native IPv4 user subscription chain in broadband, (ii) 3GPP mobile network services, and (iii) access to application level SFC in data center.



(a) SFC in broadband network

(b) SFC in mobile network



(c) SFC in data center

Figure 1.4: Service Function Chains (SFCs) in different domains

The process of sequencing an in-network middlebox chain is commonly referred to as *SFC Orchestration* [42]. Currently, middleboxes are attached to particular routers or switches within the network. In most cases, operators route traffic through the required sequence of middleboxes by manually crafting the routing table entries [43]. Moreover, the proprietary and closed-source nature of these middleboxes creates isolated silos of management and operational groups that result in undesirable hindrance in the SFC orchestration process. Network operators need to coordinate between multiple management groups for resource allocation and routing table manipulation for setting up a service chain. Middleboxes effectively behave as bumps on the wire where the traffic leaves the primary forwarding path to go through the middlebox and then return to the primary path again. It is a cumbersome and error-prone process that introduces a lot of management and operational headache. Furthermore, the fixed locations of middleboxes forces the network operator to use suboptimal routing paths. Network flows are negatively impacted in this architecture; they need to travel to particular middleboxes to get the required services. If we could instantiate a middlebox in real-time, at locations where they are required, at a particular time, then we will be able to better serve network flows.

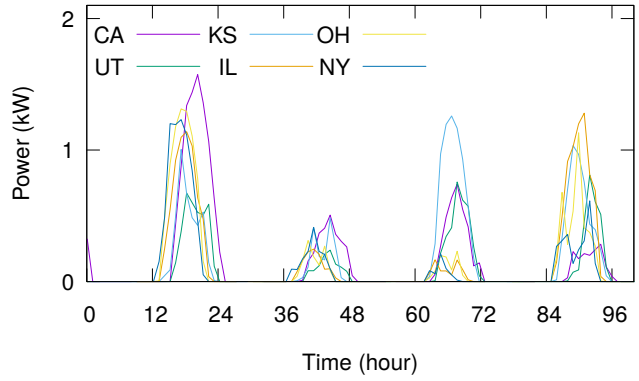
Network Function Virtualization or NFV is an emerging and promising technology that can address these limitations [44, 42, 11]. It proposes to move packet processing from hardware middleboxes to software programs or Virtual Network Functions (VNFs) running on commodity (*e.g.*, `x86` based systems) servers. This approach decouples the packet processing logic from the underlying hardware appliance and brings the benefits of economies of scale, similar to that in cloud computing, to the network infrastructure. NFV provides ample opportunities for network optimization and cost reduction. Previously, middleboxes were placed at fixed locations, but now we can deploy a VNF at any location where there are compute resources. In telecommunication networks, these locations are typically called Central Office (CO) or Point-of-Presence (PoP), and they already contain the required infrastructural setup for compute and network resources that can be utilized for VNF deployment. NFV can open-up these resources to enable opportunistic utilization of the network infrastructure. Efficient algorithms can be designed to simultaneously optimize VNF locations and traffic routing paths that will significantly reduce the network OPEX. This approach will not hamper performance as many state-of-the-art software middleboxes have already shown the potential to achieve near-hardware performance [45, 46, 47].

Another interesting research challenge in service function chaining, is the issue of resource orchestration for energy efficiency. Most often middleboxes are provisioned to handle the peak traffic of a network and consume significant amount of energy. For example, in 2013, worldwide telecommunication networks consumed around 83 Giga-Watts of electricity that is equivalent to the power consumed by 12 New York cities [48]. The huge energy

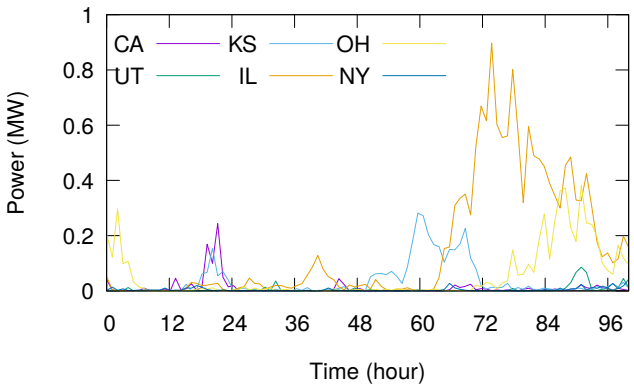
requirement comes with negative environmental implications through emission of several million tons of carbon [49]. In 2007, the annual carbon footprint for the Information and Communication Technology (ICT) sector was around 830 metric tons and is projected to double by 2020 [50]. Within the ICT sector, the telecommunication industry is responsible for around 47% of the total carbon emissions [50]. With the growing demand for today's bandwidth-intensive Internet services, the energy requirement of these networks will continue to grow and increase their carbon footprint. Several government regulations and carbon emission taxes were established to drive companies to use renewable sources of energy [51, 52, 53]. Moreover, an organization's reputation can significantly decrease if it has a high carbon footprint or does not disclose carbon emission values [54]. Hence, to comply with regulations and uphold corporate reputation, telecommunication networks need to reduce carbon footprint without significantly increasing operational costs [55, 56, 57].

Network softwarization enables innovative ways to reduce the carbon footprint of a network. For example, the utilization of on-site renewable energy sources like wind turbines or solar panels can be maximized by migrating service chains across COs or PoPs. SFCs can be provisioned intelligently in a topology aware manner to opportunistically switch-off unused servers, switches, and switch ports [58, 59]. However, the availability of renewable energy sources like solar and wind fluctuates a lot between different locations. They are very intermittent in nature and vary with time and weather conditions. Using the data collected by the U.S. Climate Reference Network and Regional Climate Reference Network [60], estimated energy generation potential from solar and wind for six different cities in six different states in the U.S. are shown in Figure 1.5(a) and Figure 1.5(b), respectively. As we can see from the figure, the production rate of renewable energy fluctuates both in spatial and temporal dimensions significantly. Intelligent and adaptable resource orchestration schemes are required to handle this fluctuation and efficiently utilize the available renewable energy.

With SDN and NFV, VNFs are not restricted to any fixed location; they can be provisioned on any compute server within the network. Moreover, coupled with an SDN controller, VNFs and their associated traffic flows can be migrated to different compute servers within a short time-frame [61, 62, 63]. Intelligent algorithms can be designed to reduce the overall energy consumption and carbon footprint by (i) opportunistically utilizing more resources at locations with surplus renewable energy while minimizing consumption at locations where brown energy is the only option and (ii) making VNF placement decisions in a manner that allows switches, switch ports, and servers to be put into low consumption state to reduce the overall power consumption.



(a) Solar energy



(b) Wind energy

Figure 1.5: Fluctuation in renewable energy production

1.3 Research Contributions

This dissertation makes the following contributions:

1.3.1 Dynamic Controller Provisioning

SDN has emerged as a new paradigm that offers programmability required to dynamically configure and control a network. SDN decouples the data and control planes and relies on a centralized server to implement the control plane. However, in large-scale WAN and data centers, a single physical server will have several performance and scalability limitations. To address these issues, multiple controllers are typically deployed to work cooperatively to control the switches in the network. Nonetheless, this approach raises an interesting problem, which we call the Dynamic Controller Provisioning Problem (DCPP). DCPP requires dynamic scaling of the number of controllers and their locations with changing network conditions, in order to minimize flow-setup time and communication overhead for state synchronization between controllers. To tackle this problem, we develop a system for deploying multiple controllers in a network. Our system dynamically scales-up or down the number of active controllers and delegates to each controller a subset of OpenFlow switches according to network dynamics while ensuring minimal flow-setup time and communication overhead. To this end, we formulate the DCPP as an Integer Linear Program (ILP) and propose two heuristics to solve it. Simulation results show that our solution minimizes flow-setup time while incurring very low communication overhead.

1.3.2 Orchestrating Virtual Network Functions

Middleboxes or network appliances like firewalls, proxies and WAN optimizers have become an integral part of today's ISP, data center, and enterprise networks. Middlebox functionalities are usually deployed on expensive and proprietary hardware that require specially trained personnel for deployment and maintenance. Middleboxes contribute significantly to a network's capital and operation costs. In addition, organizations often require their traffic to pass through a specific sequence of middleboxes for compliance with security and performance policies, which leads to complicated routing configurations for the network. NFV is an emerging and promising technology that is envisioned to overcome these challenges. It proposes to move packet processing from dedicated hardware middleboxes to software running on commodity servers. However, in NFV, it is a challenging problem to determine the required number and placement of VNFs that optimizes network operational costs and

utilization, without violating service level agreements. We call this the VNF Orchestration Problem (VNF-OP) and provide an Integer Linear Programming (ILP) formulation. We also provide a dynamic programming based heuristic to solve larger instances of VNF-OP. Trace driven simulations on real-world network topologies demonstrate that the heuristic can provide solutions that are within 1.3 times of the optimal solution. Our experimental results suggest that deploying VNFs instead of hardware middleboxes can provide more than four times reduction in the operational cost of a network.

1.3.3 Energy Smart Service Function Chain Orchestration

Telecommunication COs and PoPs are deployed in cities and metropolitan areas to provide broadband and wireless Internet services to both commercial and residential customers. These locations host a large number of middleboxes to provide last mile connectivity and different network services [64]. In recent years, the rapid development of technologies like cloud computing, photo-intensive social networks, on-demand video streaming, online gaming, and the Internet of Things (IoT) has triggered a tremendous growth in the telecommunication industry. With the growing demand for today's bandwidth-intensive Internet services, the energy requirement of these networks will continue to grow and increase their carbon footprint. In this context, we propose an Energy Smart Service Function Chain Orchestrator called *ESSO* that reduces the overall carbon footprint of a telecommunication network by opportunistically adapting the SFC embedding locations to utilize more energy at locations with surplus renewable energy. ESSO minimizes brown energy consumption by migrating SFCs across different locations. In addition, ESSO provisions individual VNFs and the virtual links between them in a manner that allows switches, switch ports, and servers to be put into low-power consumption state. ESSO employs a number of heuristic algorithms to take embedding, consolidation, and migration decisions to utilize a maximal amount of renewable energy and reduce the overall carbon footprint of the network.

1.4 Thesis Organization

The rest of this thesis is organized as follows: The work on dynamic controller provisioning is discussed in Chapter 2. Next, we describe the issue of VNF orchestration in Chapter 3. After that we present the work on energy smart SFC orchestration in Chapter 4. Concluding remarks are presented in Chapter 5 including a summary of the contributions of this thesis and a discussion of potential future research directions.

Chapter 2

Dynamic Controller Provisioning

Software-defined networking (SDN) has emerged as a new paradigm that offers the programmability required to configure and manage the network dynamically. By separating the control plane from the data plane and shifting the control plane to a logically centralized controller, SDN provides network operators the flexibility to implement a wide-range of network policies (*e.g.*, routing, security, fault-tolerance) and the ability to rapidly deploy new network technologies.

The most common SDN implementation today relies on a logically centralized controller that possesses a global view of the network. Whenever a switch receives a new flow, it requests the controller to install appropriate forwarding rules along the desired flow path. The time required to complete this operation is known as the flow-setup time. However, in a large-scale WAN deployment, this rudimentary centralized approach has several limitations related to performance and scalability. First, it is not always possible to find an optimal provisioning of the controller that can ensure acceptable latencies between the controller and the switches situated at geographically distributed locations. Secondly, a single controller usually has a limited resource capacity and hence cannot handle a large number of flows originating from the infrastructure switches. In this case, the average flow-setup time can rise significantly and degrade application and service performance [28].

To address these limitations, recent proposals have advocated to deploy multiple controllers that work cooperatively to better manage network traffic flows [24, 22]. Nonetheless, this approach introduces a new problem: minimizing flow-setup times by dynamically adapting the number of controllers and their locations according to demand fluctuations in the network. We call this problem the Dynamic Controller Provisioning Problem (DCPP). Specifically, DCPP requires the number of controllers to be sufficient to handle the current

network traffic, and their locations should ensure low switch-to-controller latencies. However, the multi-controller deployment also requires regular state synchronization between the controllers to maintain a consistent view of the network [19]. This communication overhead can be significant if the number of controllers in the network is large. Finally, as network traffic patterns and volumes at different locations can vary significantly over-time, the controller provisioning scheme has to react to network "hotspots" and dynamically re-adjust the number and the location of controllers. Hence, the solution to DCPP must always find the right trade-off between performance (in terms of flow-setup time) and overhead (controller synchronization and management).

To the best of our knowledge, the only work that has investigated the controller provisioning problem is the one by Heller et al. [15]. They studied a static version of the problem where controller provisioning remains fixed over time and analyzed the impact of the controller locations on the average and worst-case controller-to-switch propagation delay. However, a static controller provisioning configuration may not be suitable for very long as network conditions can change over time.

To address this limitation, we propose a management system for dynamically deploying multiple controllers within a WAN. Specifically, we consider the dynamic version of the controller provisioning problem where both the numbers and locations of controllers are adjusted according to network dynamics. Our solution takes into account the dynamics of traffic patterns in the network while minimizing costs for (i) switch state collection, (ii) inter-controller synchronization, and (iii) switch-to-controller reassignment. First, we provide background on SDN and its control plane in Section 2.1 and Section 2.3, respectively. After that, we describe our proposed management system in Section 2.4. Next, we formulate DCPP as an Integer Linear Program (ILP) that considers all aforementioned costs in Section 2.5. We then propose two heuristics that dynamically estimate the number of controllers and decide their placement in order to achieve the desired objectives in Section 2.6. The effectiveness of our solution is then demonstrated using realistic traces and WAN topologies in Section 2.7. Our results show that the proposed algorithms minimize the average flow-setup time while incurring very low communication overhead. Related work is presented in Section 2.2, and finally, we provide concluding remarks in Section 2.8.

2.1 Software-Defined Networking (SDN)

The Open Networking Foundation (ONF) defines Software-Defined Networking or SDN as an emerging networking architecture that decouples network control from forwarding in the data plane, enables direct programmability of the network, and abstracts the underlying

infrastructure for network applications and services [65]. SDN provides open interfaces to program and control network elements (*e.g.*, switch or router) to define the data forwarding and processing operations. SDN separates the control and data forwarding functions and relocates the control functions into a dedicated entity called the SDN Controller. The behavior of the network elements is controlled and managed by SDN *Applications* that run on top of the controller. Another differentiating feature of SDN is the treatment of traffic in terms of flows based on protocol headers of a packet. A flow in SDN represents all packets in a data flow that have the same set of values for a specified set of protocol header fields.

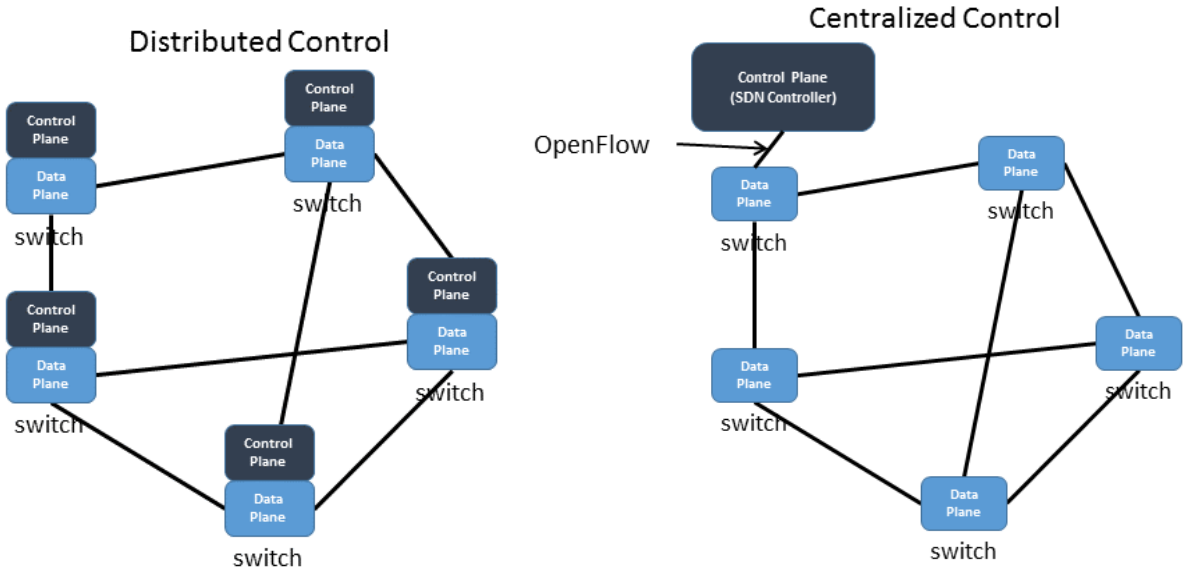
2.1.1 Traditional vs. Software-Defined Network

Traditional IP networks are operated by running distributed routing protocols that are embedded within the router or switch. These routing protocols exchange messages with neighboring devices to converge to a consistent view of the network and then determine traffic forwarding rules by executing path computation algorithms. Each device constructs its own perspective of the network, and the only way to inspect this view is by connecting to the device via its CLI. In a traditional network, the data and control planes are blended within the device as shown in Figure 2.1(a). Network operators express high-level policies by manually configuring each device through the device's CLI or other vendor-specific tools, which is a complicated, cumbersome, and error-prone process. Moreover, due to the closed-source, vertically-integrated, and proprietary nature of network devices, automatic reconfiguration and dynamic adaptation to traffic fluctuations are very challenging in current networks.

SDN proposed to address this issues by decoupling the data and control planes. The SDN architecture enables the control and data planes to evolve independently. It facilitates isolated optimization and life-cycle management of data and control plane technologies. Figure 2.1(b) depicts an SDN network where the control plane of the network devices are placed on a logically centralized and programmable software platform known as the network controller.

2.1.2 SDN Architecture

The SDN architecture, proposed by ONF, is shown in Figure 2.2 [65]. ONF divides the SDN architecture into three layers: (i) Infrastructure Layer, (ii) Control Layer, and (iii) Application Layer. The infrastructure layer consists of the data plane devices like switches



(a) Traditional network

(b) Software-defined network

Figure 2.1: Routing in traditional vs. software-defined networks

and routers. This plane is commonly referred to as the data plane. Its primary functions include data forwarding, local monitoring, and statistics collection. The control layer includes the SDN control software or the network controller; it can be deployed either on a single server or cluster of servers to satisfy performance, reliability, security, and availability requirements.

The separation between the control and data planes is achieved by means of a well-defined and standardized communication channel between the controller and data plane devices. The protocol for the communication between the controller and data plane is referred to as the Southbound API (SBI). OpenFlow has emerged as the de-facto standard for SBI by providing flow-level granularity, vendor-neutrality, and an abstraction that is generic enough to promote innovation and provide adequate functionality for network programmability. Key control plane functions include the control and monitoring of the network infrastructure. Data plane devices expose their functionalities, capabilities, and current state to the control plane through a specific set of messages defined by the SBI. The controller collects network statistics from network devices using these messages and maintains a global view of the network. The control plane also provides the means to

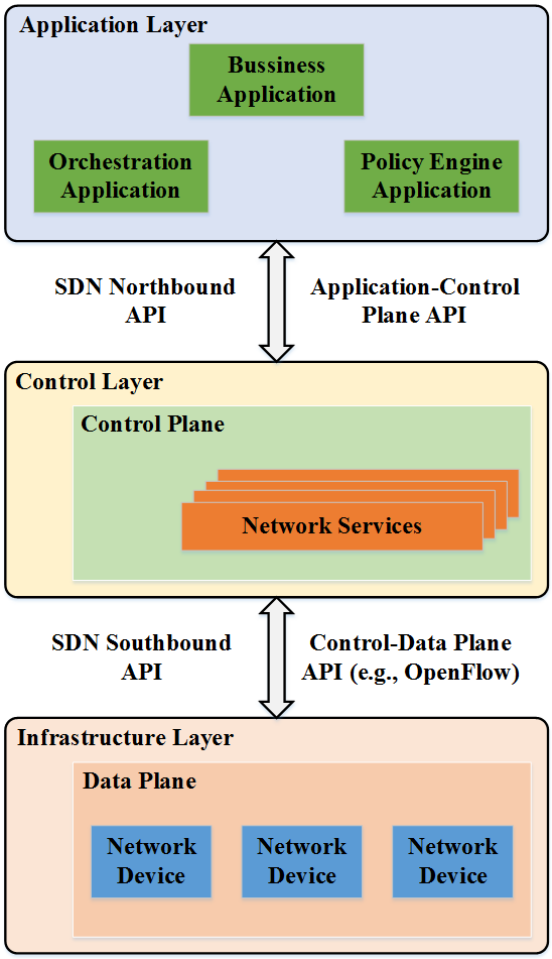


Figure 2.2: SDN architecture

control data plane devices through a set of SBI messages.

The controller provides the global network view to the application plane through a standardized and programmable API known as the Northbound API (NBI). The application layer consists of applications or services like router, firewall, and load-balancer, implemented as special purpose software modules that run on top of the controller. These applications use the NBI to program network devices through the controller. Unlike the SBI, there is still no standardized NBI; each controller implementation provides its own feature-set and implementation for the NBI. The controller translates the application logic to control instructions (defined by SBI) for the network devices enabling infrastructure

independent development of SDN applications. The behavior of a switch or router is governed by the instructions or flow-table rules received from the controller that are sanctioned through the policies imposed by the applications running on top of it.

2.1.3 OpenFlow Switch

An OpenFlow enabled switch is shown in Figure 2.3. It has one or more flow-tables that contain forwarding rules to determine how to handle data traffic. Each rule matches a subset of fields in the packet header; it identifies a flow in the network and can perform operations like drop packet, modify packet headers, or forward packet through one or more outgoing ports. The group table contains flow-entries to perform operations like flooding, aggregation, and multi-path forwarding. The meter table is used to control parameters related to QoS of the switch. Each switch also contains one or more OpenFlow channel or agent to communicate with the controller. The communication is usually performed over a secure channel to exchange both control and data packets. The controller can add, update, or delete flow-table entries in the flow-tables via the OpenFlow protocol.

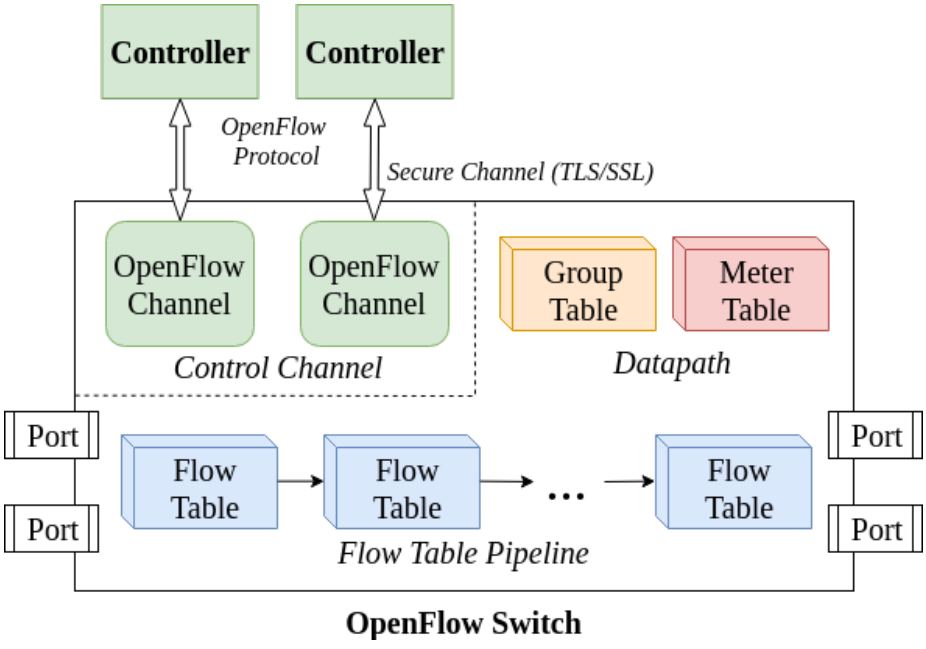


Figure 2.3: Components of an OpenFlow switch

Typical components of an OpenFlow flow-entry are depicted in Figure 2.4. The match

field contains the ingress port, packet headers, and other metadata to match incoming packets (Figure 2.5). Match fields can contain wildcards to match aggregate flows. A switch typically uses Ternary Content Addressable Memory (TCAM) to enable fast lookups for wildcard matches. The priority field in the flow-entry represents the matching preference within a flow-table, counters are used for storing match statistics, instructions are used to control processing of packets through the flow-tables, timeout is used to expire a flow-entry, and the cookie is used by the controller to filter flow-entries. The counters store the number of received packets and bytes, which can be used to determine different metrics about the links connected to the switch. The OpenFlow protocol provides packet and byte counters at the granularity of flow-table, flow-entry, port, and queue.

Match Fields	Priority	Counters	Instructions	Timeout	Cookie

Figure 2.4: Components of an OpenFlow flow-entry

Ingress Port	Ethernet			VLAN		IP				TCP/UDP	
	SA	DA	Type	ID	Priority	SA	DA	Proto	TOS	SRC	DST

Figure 2.5: Match fields in OpenFlow flow-entry

Each OpenFlow switch contains one or more flow-tables. Packet matching always starts at the first flow-table, then based on the instructions installed in the matching flow-entries it can be forwarded to other flow-tables in the flow-table pipeline (see Figure 2.3). Flow-entries are matched according to the priority (higher value means higher priority) of the flow-entries in the flow-table. The pipeline processing is performed in two stages: (i) ingress and (ii) egress processing. If a matching entry is found, then the processing ends with either dropping, modifying and/or forwarding the packet through one or more output port. If no entry matches the flow, then the outcome depends on the configuration of the table-miss flow entry, which has the lowest priority and a wild-card match rule to match all packets. The table-miss flow-entry is usually configured to forward packets to the controller; however, it can also drop all packets or process packets through the traditional (non-OpenFlow) pipeline of the switch.

OpenFlow uses a special output port called '`NORMAL`' to represent the traditional processing pipeline of the switch. Any flow-entry with an action to forward a packet to the `NORMAL` output port, skips the OpenFlow based flow-table pipeline and hands-over the packet to the traditional fully-distributed routing protocol based forwarding mechanism of

the switch. In this case, the forwarding is performed based on the information gathered in Routing Information Base (RIB) of the switch using routing protocols like OSPF, STP, or RIP.

2.1.4 Switch – Controller Interaction

The process of handling packets for which no flow-entry is installed in the switch is depicted in Figure 2.6 and Figure 2.7. In the traditional OpenFlow switch configuration, if the packet-miss flow entry specifies that a packet should be forwarded to the controller, then every first packet of a new incoming flow (without a matching flow-entry) is forwarded to the controller (step ① in Figure 2.7). The packet is either buffered at the switch or encapsulated in the packet forwarded to the controller based on available buffer-space at the switch (step ②). In either case, the packet headers are included in the special control packet called `packet_in`, which is sent from the switch to the controller.

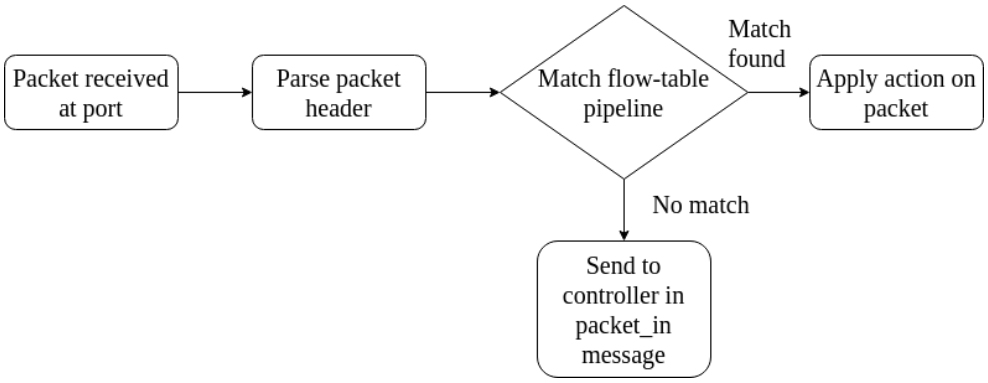


Figure 2.6: Packet forwarding by an OpenFlow switch

The controller extracts the data packet information from the `packet_in` message and forwards the packet to a controller application responsible for handling such `packet_in` messages. The application eventually determines one or more flow-table entries to handle the corresponding flow. The controller then sends back another control packet, called the `flow_mod`, to install the new rules in the switch's flow-table (step ③ in Figure 2.7). Subsequent packets in the flow are processed by the newly installed flow-table entries. In most cases, the controller sets up the entire path by installing flow-entries in all switches on the path used to forward the flow packets, as shown in step ④ of Figure 2.7.

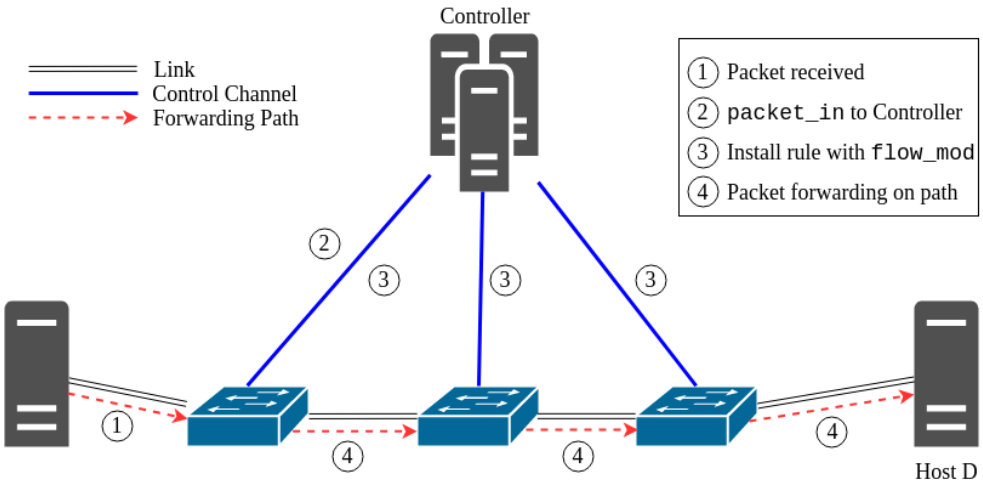


Figure 2.7: Path setup for new flow

2.2 Related Work

2.2.1 Control Plane Scalability

SDN's proposal to have a logically centralized control plane has motivated a spate of research works that explored the scalability, performance, and reliability of the control plane under different design assumptions. The first and basic design choice is the case of a single controller. Heller *et al.* was the first to analyze control plane performance based on only switch to controller latencies [15]. They analyzed a number of network topologies and showed that the number and placement of controllers depend on the desired controller to switch reaction time, the metric of choice to be optimized, and the network topology. They also reported that for the Internet2 [66] topology that has 34 nodes and 41 links, at least three controllers are required to ensure a reasonable controller to switch latency [15]. However, in their work, Heller *et al.* did not consider either the effect of load on the controller or the variation of traffic load across time and location. Tootoonchian *et al.* tested the performance of an SDN controller called NOX [67] in [28]. They developed a tool called *cbench* to stress test SDN controllers, and reported that if the number of switches is far greater than the number of available CPU cores, then the average response time of the controller increases significantly. Increased contention across threads, TCP dynamics, and internal task scheduling overhead at the controller are the primary reasons for this performance degradation. A number of research works have proposed different techniques to overcome this scalability limitation. These techniques can be classified into

two broad categories: (1) pushing intelligence into the switch to offload the controller, and (2) distributing the control plane across multiple controllers.

Pushing Down Intelligence

DevoFlow [18] and DIFANE [17] fall in the first category of techniques. DevoFlow proposes to pre-install wildcard rules in the switches that can replicate themselves for the mice flows to create flow specific rules. The switches also have the intelligence to detect elephant flows. The controller is only responsible for making the forwarding decision for elephant flows. Similarly, in DIFANE, the controller generates the forwarding rules, but it is not involved in the setup of each new flow. Rather, the rules are partitioned and distributed among a subset of switches called "authoritative switches". The regular switches, which forward packets in the data plane, redirect new flows to the authoritative switches to learn about the forwarding rules. However, both of these proposals require some changes to be made to the commodity switches to increase their intelligence.

Distributed Control Plane

On the other hand, Kandoo [22], HyperFlow [24], and Onix [21] propose to distribute the control plane across multiple controllers to improve SDN's scalability. Each of them distributes controller states differently. Kandoo distributes controller states by placing the controllers in a two-level hierarchy comprising a root controller and multiple local controllers. Local controllers respond to the events that do not depend on global network state (*e.g.*, elephant flow detection), while the root controller takes actions that require global network view (*e.g.*, re-routing elephant flows). Hyperflow provides a logically centralized view of the control plane to network applications. However, the control plane itself is physically distributed. Each physically distinct controller in the network is responsible for handling requests from a subset of the network switches. HyperFlow handles state distribution of the distributed controllers through a publish/subscribe system based on the WheelFS [68] distributed file system. The physically distributed controllers share the same network-wide view through the publish/subscribe system. Finally, controller state distribution in Onix is managed through a distributed hash table. This allows the controllers to handle network events locally without the necessity of communicating with remote nodes.

2.2.2 Multi-Controller Provisioning

None of the aforementioned works considered the issue of choosing suitable network locations for controller provisioning and adapting the provisioning according to the dynamic behavior of the network. To the best of our knowledge, we were the first to identify this problem as the Dynamic Controller Provisioning Problem (DCPP) [69]. We also proposed a management system that takes both network performance (in terms of flow-setup time) and management overhead (for state synchronization) into consideration to determine the number and placement of controllers in the network. A good number of research works built upon our work on multi-controller provisioning. Yao *et al.* considers a variation of DCPP where different controllers have different processing capacities and allocated switches in a way that does not overload any controller in the network [70]. While in our experiments we used controllers with the same capacity, there are no restrictions in the problem formulation or the heuristic algorithm on using a heterogeneous class of controllers.

Another work that followed is the Pareto-Optimal Controller Placement (POCO) framework [71, 72] that considers additional metrics like controller-controller latency, load-balancing among controllers, and resilience in case of double node or link failures. The authors propose an algorithm that generates all possible combinations for k-controllers and then selects the best one based on the metrics mentioned above. The number of controllers (k) is an input to the algorithms. In contrast, we determine the optimal number of controllers and also change their number and location based on traffic fluctuations. Lange *et al.* extended the algorithms in POCO, with heuristics so that it can support larger and dynamic networks. However, the number of controllers is still taken as an input, instead of being optimized. Obadia *et al.* proposed a greedy heuristic to assign switches to a controller in a manner that minimizes control plane traffic [73]; whereas, our work minimizes both control plane traffic and flow-setup time. Rath *et al.* proposed a zero-sum game based distributed controller provisioning mechanism where a controller communicates with its neighboring controllers to determine a payoff for staying active. However, the authors did not provide any details on the control plane messaging overhead.

Another group of works targets the controller provisioning problem from the viewpoint of reliability. Hu *et al.* introduce the reliability-aware controller provisioning problem and propose a metric called *control-path loss* to measure the reliability of a controller placement [74, 75]. This group of works targets a different problem than DCPP; they are more similar to fault-tolerant virtual network embedding problems [74]. Another related work is presented in [76], where the authors provide a mathematical formulation and heuristic algorithms for ensuring five-nines availability. However, in their work, they assume that a switch can connect to more that one controller simultaneously and send updates regarding

its state to all of the controllers. While the first assumption is in line with traditional SDN switch-to-controller interaction, the second one will need modifications to the OpenFlow standard. Authors in [77, 75, 78] proposed algorithms that provision controllers to ensure that the control network can tolerate single link failures.

2.3 Control Plane Assumptions

2.3.1 Logically Centralized – Physically Distributed

SDN control plane is typically deployed as a logically centralized and physically distributed collection of servers, as shown in Figure 2.8. A single controller faces problems and issues related to scalability, reliability, performance, and single-point-of-failure. For this reason, a cluster of servers at a location or multiple servers distributed across locations are generally used to implement the SDN control plane. In Figure 2.8, each controller is in-charge for a subset of the switches. A controller collects network statistics from the switches which are under its control and then synchronizes the collected data with other controllers using the Controller-to-Controller API (C2I), which is also known as the East-West API.

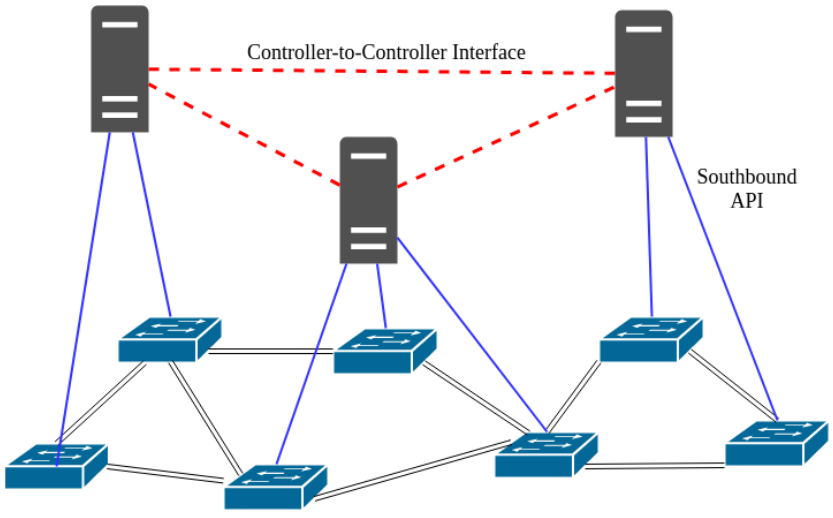
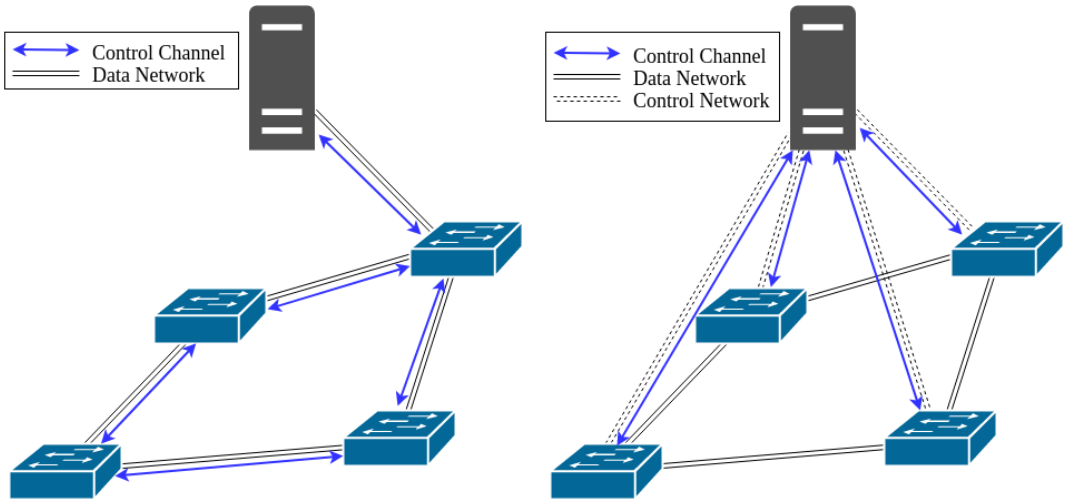


Figure 2.8: Control plane deployment

The C2I is not yet standardized; different controller vendors implement different interfaces for inter-controller communication. However, irrespective of the vendor, all SDN

controllers provide a global network view to the network applications. The distribution of control plane responsibilities can be organized in either a flat or hierarchical manner. A flat structure is similar to the one shown in Figure 2.8, where each controller is responsible for a subset of switches, and the controllers communicate with each other in a peer-to-peer manner to synchronize data. In a hierarchical structure, controllers are organized into layers of nodes in a tree. Controllers on the higher layer manage controllers on a lower layer. Only the controllers on the lowest layer interact directly with the switches, and their interactions are governed by the policies enforced by the controller at the higher layers.

2.3.2 In-Band vs. Out-of-Band Signaling



(a) In-band control (b) Our-of-band control

Figure 2.9: In-band vs. out-of-band control channel

The communication between the controllers and switches can be either in-band or out-of-band. With in-band control, control traffic shares the same network infrastructure as data traffic, as shown in Figure 2.9(a). This approach does not require a separate network for control traffic; however, it has security and fault-tolerance issues. Control traffic is passed through the same network as data traffic, which raises security concerns. In addition, failure of any link in the data network can affect the control channel as well. Out-of-band control is an alternative to overcome this issues. The out-of-band model for control plane signaling is shown in Figure 2.9(b). Out-of-band control requires a separate network

between the controllers and the switches, which increases both capital and operational cost of the network. Moreover, in most cases, out-of-band control is not a feasible solution for ISPs and carrier wide-area networks, as these networks can span multiple countries or continents. A separate physical network may not be feasible or cost-effective. Out-of-band control is commonly used for data center networks, where a separate network is usually deployed for device management.

2.3.3 Control Channel Bootstrapping

Control channel bootstrapping refers to the setup of initial paths for switch-to-controller and controller-to-controller communication. These paths are essential for the proper operation of an SDN, as without such paths the switches will not be able to connect with the controller, and the controllers will be able to set up rules in the switches for data traffic forwarding. An SDN network can operate only after all required paths between controller and switches are configured. In case of out-of-band control, the control network typically uses traditional routing protocols like STP or OSPF to set up these paths, and the data network is operated based on the flow-table rules installed by the controller. However, for in-band control, the flow-tables of the switches need to contain rules for forwarding both control and data traffic. In addition, operating both traditional and software-defined forwarding protocols for the same network increases the management complexity of the network [79]. Hence, most SDN networks utilize a bootstrapping phase to set up the initial paths. Different approaches have been proposed in the literature to learn the topology information used to bootstrap an SDN network [80, 81]. In our work, we also assume the existence of a bootstrapping entity other than the controllers used to operate the network, which is described in detail in Section 2.4.

2.3.4 Switch Reassignment

In this work, we dynamically scale up and down the active controller pool based on the temporal and spatial variation in traffic load of the network. However, to ensure balanced load distribution among the controllers, we need to reassign or migrate switches between controllers. The mechanism to achieve switch reassignment requires coordination between the source and target controllers and the switch. A number of research works have proposed switch reassignment or migration mechanisms for this purpose [82, 83, 84]. As our primary focus in this work is the determination of the optimal number and locations of controllers, we adopt the switch migration protocol outlined in [83].

Authors in [83] propose a four-phase disruption-free switch migration protocol. *Phase-I* utilizes the "equal mode" semantics provided by the OpenFlow protocol. In OpenFlow, a switch typically has a single master and multiple slave controllers [85]. The master controller has write access to switch configurations, while the slave controllers have read-only access. However, in equal mode, multiple controllers can have write access to the switch. In phase-I, the target controller is put into the equal mode so that it will receive notifications for all events just like the master controller. In *phase-II*, the current source or master controller installs a dummy, but pre-specified flow-entry in the switch. It then sends a barrier request message to force the switch to process all pending events. Upon the processing of all pending requests at the switch, the master controller removes the dummy flow-entry, which triggers a flow-removed event and this message is sent to both the source and target controllers. This event marks the handover of the switch from the source to target controller. In *phase-III*, the source controller performs another barrier request-reply cycle to process any buffered events that might have occurred during phase-II and sends an end-of-migration message to the target controller. Next, in *phase-IV*, the target controller sends a role-request message to the switch to become the new master controller for that switch. All subsequent events originating from the switch are processed by the target (new master) controller from this point. Through this mechanism, a switch is migrated from one controller to another whenever switch-to-controller mapping is updated.

2.3.5 Inter-Controller Communication & State Synchronization

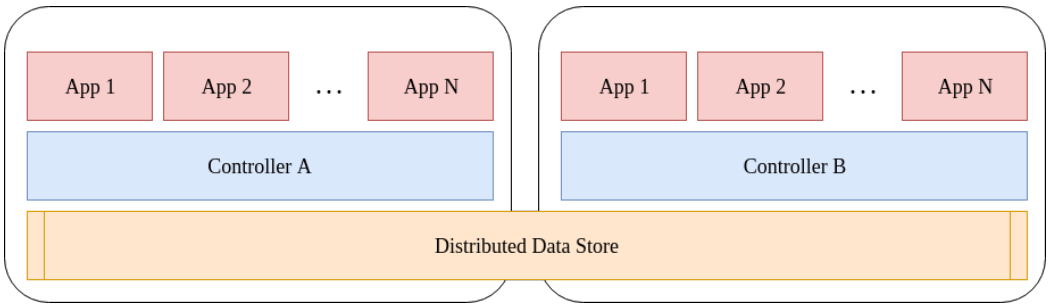


Figure 2.10: Distributed data-store for inter-controller state synchronization

A multi-controller setting requires state synchronization between the controllers both for maintaining the global network view and ensuring consistent network updates from the network applications running on top of the control plane. This work assumes that all switch and topology specific information is saved in a distributed key-value store (*e.g.*,

Cassandra) as shown in Figure 2.10. The global network view might include information like topology, network devices, device capabilities, reachability between nodes, link latency; bandwidth; packet-loss rate, and dynamic state like flow-table, flow-entry counters, and link utilization. Each controller reads network topology and switch related statistics from the distributed data store. Whenever there is a change within a controllers domain, the controller updates the relevant information in the distributed data store. We assume that each controller participates as a node in the distributed data store, so that network related data is obtained locally. Similar assumptions are made by Onix [21], which is a distributed OpenFlow controller that uses a Distributed Hash Table (DHT) to store the global network view. Each controller maintains a local view of the sub-network under its control and periodically updates the global network view in the DHT. HyperFlow [24] is a distributed OpenFlow controller that uses a distributed file system called WheelFS. Each controller is responsible for a certain local area and announces events that affect the global view of the network periodically. These events are replayed at the other controllers to achieve a synchronized global network view. OpenDaylight [8] and ONOS [86] are two other SDN controllers that support distributed control plane and provide mechanisms for state synchronization. In essence, distributed controllers utilize distributed storage technology as a backend to achieve state synchronization between multiple controllers. However, these approaches only work for a particular controller; they are not interoperable as there is no standardized inter-controller communication protocol. SDNi [87] and East-West Bridge [88] propose inter-controller messaging protocols to perform state synchronization between heterogeneous controllers. The state synchronization mechanism is orthogonal to this work, and any of these proposals can be adopted for our purpose.

2.4 System Description

In this work, we consider a moderate to large size WAN consisting of OpenFlow enabled switches and compute resources to deploy OpenFlow controllers. We propose a controller management system that works with a mix of OpenFlow and non-OpenFlow switches, where non-OpenFlow switches are configured through standard management protocols like SNMP and behave as opaque forwarding elements between OpenFlow enabled switches. We also assume that compute resources (*e.g.*, servers) are deployed at designated locations throughout the network. These servers are used to deploy OpenFlow controllers to control the OpenFlow enabled switches in the network.

In most cases, a single controller is not sufficient for moderate to large size WAN deployments. Two well known and high-performance SDN controllers NOX [67] and Maestro

[89], can process around 30K and 600K PACKET_IN requests per seconds, respectively. A multi-threaded and I/O optimized variation of NOX can process around 2M requests [28]. However, large-scale networks can generate flows at a rate that is several orders of magnitude higher than the processing capacity of a single controller. In [31], a 1500 server cluster is reported to generate 100K flows per second, and in [89] a 100 switch network is shown to generate 10M flows per second. In addition, in [83] the authors reported that a network with 100K hosts, a peak flow arrival rate of 300M (median rate between 1.5M and 10M), and a controller with 2M flow requests per second processing capability; around 1–5 controllers are required to handle the median traffic; however, around 150 controllers are required to handle the peak traffic. Hence, a dynamic controller provisioning mechanism is required to avoid resource over-provisioning. This work proposes a management system to dynamically partitions the set of OpenFlow switches into multiple *domains* (henceforth "domain" is used to specify the set of OpenFlow switches that are controlled by a controller) based on network dynamics and assigns one controller per domain. At any time instance, a switch is controlled by a single controller and each controller is responsible for setting up paths on switches in its domain.

A controller periodically collects port, flow, and table-level statistics from switches in its domain using OFPST_PORT, OFPST_FLOW, and OFPST_TABLE OpenFlow messages, respectively. Controllers calculate switch and link level information from the collected data in its domain and push this information to the distributed data store so that each controller can make forwarding decisions on its own. Each controller builds its view of the network from the information stored in the distributed data store. A switch reports to its assigned controller and relies on it to make forwarding decisions. After receiving a flow-setup request from a switch, the controller utilizes the locally stored global network view to determine forwarding decisions.

We assume that controllers are deployed on servers located at different network locations. These locations are pre-determined by the network operator based on the architecture of the network and management policies. A controller is regarded as *active* if it has at least one switch assigned to it; otherwise, it is considered *inactive*. Inactive controllers keep listening on a particular port for incoming HELLO messages from newly assigned switches and consume a minimal amount of CPU cycles. With the change in traffic volume, a given controller placement may become sub-optimal over time. In this case, it is necessary to adjust the provisioning of controllers according to traffic fluctuations, which is achieved by dynamically activating and deactivating controllers and adjusting switch–to–controller assignment in the network. As this is a non-trivial process that can incur significant reconfiguration costs for controller setup and switch reassignment, we propose a management system to dynamically scale-up and down the active controller pool in the network.

This system monitors traffic volume and state of active controllers and periodically runs the switch–to–controller assignment algorithm to optimize the number and placement of controllers based on the network condition.

2.4.1 Control Plane Management System

The management system maintains an active pool of controllers where each controller controls a non-overlapping subset of switches. It also periodically evaluates the current switch–to–controller assignment and decides whether to perform a reassignment based on the specified constraints. If a reassignment is performed, the management system also updates the switch–to–controller assignment in the network. The management system contains four modules as depicted in Fig. 2.11 and explained below:

Bootstrapping Module

In the OpenFlow paradigm, a switch must first connect with its controller, which requires the initial setup of paths between the switch and the controller. The bootstrapping module discovers the network topology and sets up these paths. This module uses Link Layer Discovery Protocol (LLDP) and Address Resolution Protocol (ARP) messages to discover network topology. It sends LLDP packets to each switch and parses the reply to learn about its neighborhood. Then incrementally learns the entire network topology by sending LLDP messages to newly discovered devices. After learning the topology, the bootstrapping module installs rules in the switches so that control traffic can be transmitted between the switch and controller.

Monitoring Module

The monitoring module keeps track of the active and inactive controllers through periodic heartbeat messages. It also collects CPU utilization statistics from the controllers. This information is used by the reassignment module to scale up or down the active controller pool dynamically. In addition, the monitoring module records the average number of flows transmitted between switch pairs in the network, which is used to estimate the number of possible flow-requests at different controllers.

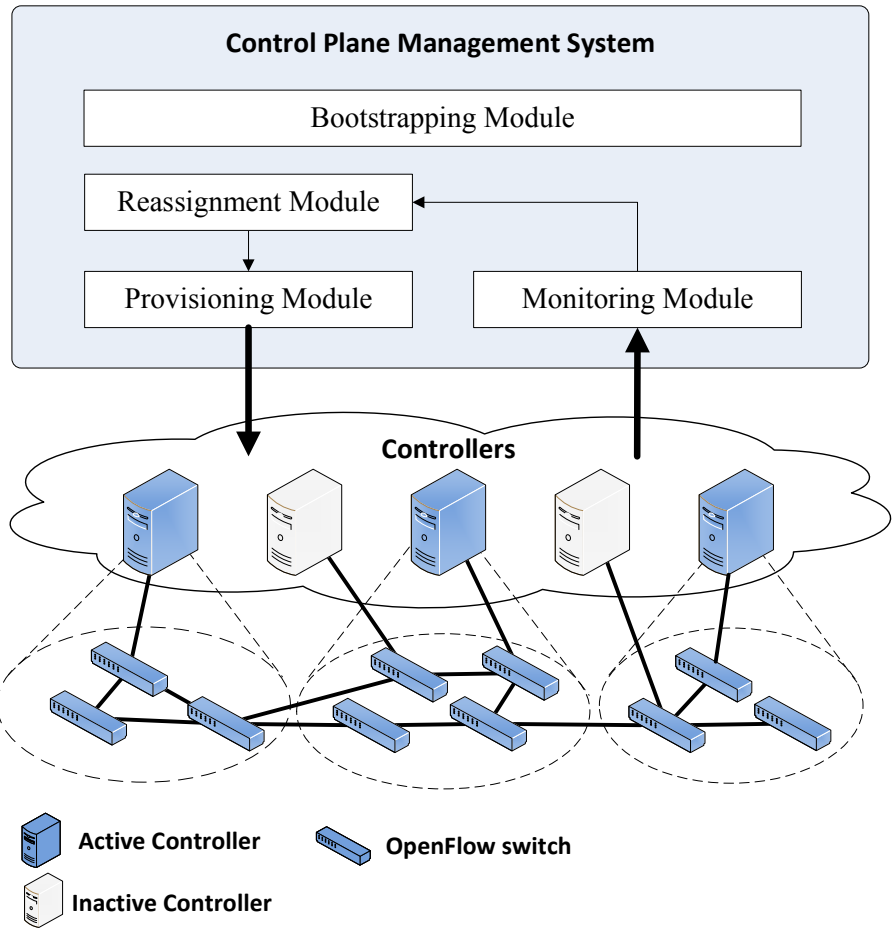


Figure 2.11: Management system architecture

Reassignment Module

This module periodically evaluates the statistics collected by the monitoring module and decides whether to perform a reassignment. This decision depends on a number of criteria that are explained in detail in the following section. The reassignment module utilizes a switch migration protocol for disruption-free switch migration between controllers during reassignment. During a reassignment, an active controller may become inactive if all of its switches are assigned to other controllers, and an inactive controller may become active if a switch is assigned to it. The objective of our system is to keep a set of controllers in the active state that results in optimal flow-setup times while incurring low communication overhead.

Provisioning Module

The provisioning module performs two actions: (i) instructs controllers to send role-change messages to switches to realize the changes in switch–to–controller mapping computed by the reassignment module and (ii) it transitions controllers between active and inactive state based on the current switch–to–controller mapping. If an inactive controller is assigned a new switch, the provisioning module instructs the inactive controller to become active and then the controller sends a role-change request to the switch to become its master controller. In a similar manner when a controller becomes inactive, *i.e.*, no switch is assigned to it, the provisioning module instructs the controller to enter inactive mode.

2.4.2 Path Setup Process

In the OpenFlow protocol, when a new flow arrives at a switch, the switch first checks its flow-table(s) for a matching entry. If a matching flow-entry exists, packets in the flow are forwarded according to the matching rule. If no such rule exists, the switch sends a `PACKET_IN` OpenFlow message to its master controller. After receiving the `PACKET_IN` message, the controller needs to compute a path using its knowledge about the network and install forwarding rules in the switches along the path. Two possible cases can arise next: (i) the flow may pass through switches only under the current controller's domain, or (ii) it may pass through switches under more than one controllers' domain. In both cases, flow-entry rules must be installed in all switches participating on the flow's path.

In the first case, the controller can directly install all required flow-entries on the switches. However, for the second case, there are two possible scenarios: (i) the controller receiving the flow-setup request from the ingress switch just installs flow-entries

on switches in its domain and does not take any action for the other switches belonging to other controller(s); then the flow is eventually forwarded to some switch in some other controller's domain, which might generate additional flow-setup requests based on the configuration of the switch, (ii) the ingress controller instructs the other controllers, involved on the path, to install the required rules in the switches under their domain. The second scenario requires modification to standard controller implementation to introduce a controller-to-controller interface for delegating flow-rule installation commands. Hence, this work adopts the first scenario as it does not require any modification to standard controller implementations.

The path setup process is shown in Figure 2.12. In this figure, a new flow arrives at switch i on port a. As the switch does not have a matching forwarding entry, it sends a flow-setup request to its controller m. This request is called *Initial path setup request*. Now, controller m computes a path (contained within its own domain) for the flow. Lets say the path is $i.a \rightarrow i.b \rightarrow j.b \rightarrow j.c$ (where $i.a$ means port a of switch i) and sets up the forwarding rules in switches i and j. The packets in the flow are forwarded from port a to port b of switch i and then from port b to port c of switch j, eventually reaching port c of switch k. Switch k now searches its forwarding table for a matching rule. If no such rule exists, it sends a flow-setup request to its controller n. This request is termed as *Intermediate path setup request*. Now, controller n computes a path contained in its own domain and installs forwarding rules in switch k and l in a similar manner.

2.5 Mathematical Formulation

2.5.1 Problem Definition

In this work, we identify and specify the Dynamic Controller Provisioning Problem (DCPP), whose primary objective is to minimize the flow-setup time by dynamically scaling-up and -down the poll of active controllers and adjusting their locations according to traffic fluctuations in the network. The other objectives of DCPP are to minimize inter-controller state synchronization overhead for maintaining a global network view and switch-to-controller communication overhead for collecting flow-level statistics. The DCPP takes as input the network topology, traffic matrix, controller capacities, and server locations. The output of DCPP is the optimal switch-to-controller assignments for the provided network topology and traffic matrix.

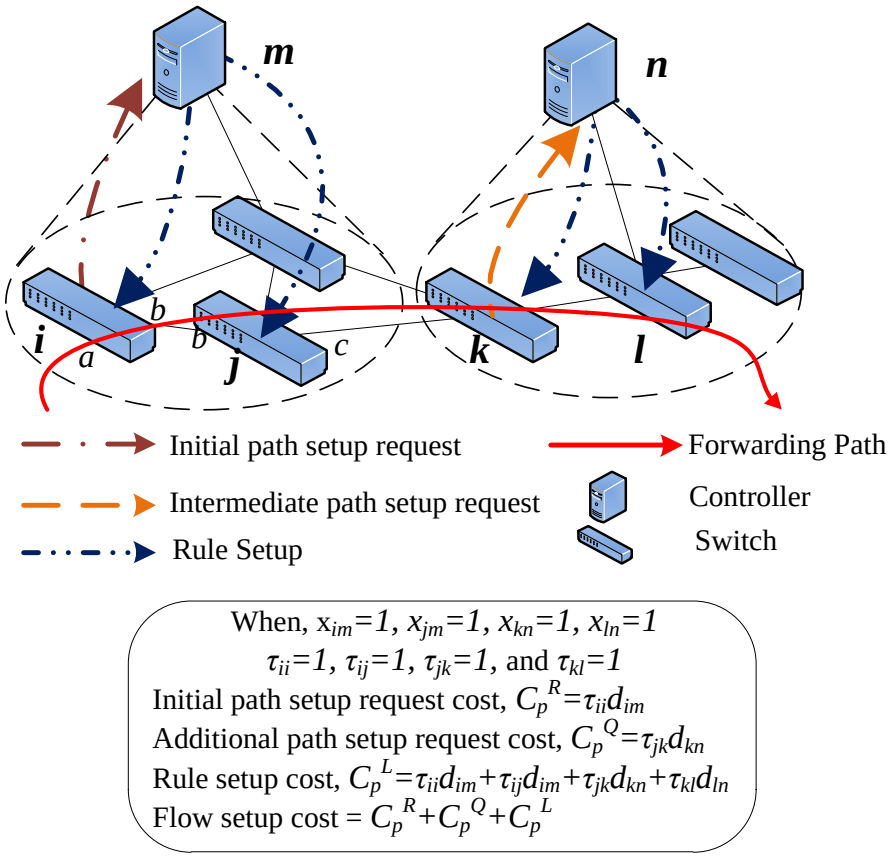


Figure 2.12: Path setup method with flow-setup cost

2.5.2 Problem Formulation

In this section, we formulate DCPP as an ILP. Specifically, we model the network as an undirected graph, $G = (S, E)$, where S is the set of switches and E is the set of edges. Let d_{ij} denote the cost of the shortest path between switches i and j expressed in terms of propagation delay or the number of hops. F ($F \subseteq S$) is the set of locations where a controller can be deployed. Let vector $U = \langle u_1, u_2, \ldots u_{|F|} \rangle$ represents the capacities of the controllers. Hence, u_m is the maximum number of requests controller m can handle per second. The maximum allowable cost between a switch and its controller is denoted by δ (expressed in the same unit as d_{ij}).

The traffic matrix is denoted by $\mathcal{T} = [\tau_{ij}]_{|S|\times|S|}$, where τ_{ij} represents the average number of flows over the current time slot originating from switch i to switch j as reported by the monitoring module. Moreover, each diagonal entry τ_{ii} of $\mathcal{T}$ captures the average number of flows originated at switch i, *i.e.*, the average number of flows coming from the networks served by switch i.

The reassignment algorithm is invoked at every T_a time interval. The output of our ILP is an assignment matrix $\mathcal{X} = [x_{im}]_{|S|\times|F|}$, where x_{im} is equal to 1 if switch i is assigned to controller m, and 0 otherwise. It also provides a binary vector $Y = \langle y_1, y_2, \ldots y_{|F|} \rangle$ indicating which controllers are active (*i.e.*, $y_m = 1$) and which are not (*i.e.*, $y_m = 0$).

We consider the following four costs that will be incurred when deploying multiple controllers across the network:

1) **Statistics collection cost** (C_l) is the number of messages per second required for the controllers to collect statistics from their associated switches. Assuming that statistics are gathered at each time interval T_s (note that $T_s < T_a$), this cost can be expressed as follows:

$$C_l = \left\lfloor \frac{T_a}{T_s} \right\rfloor \sum_{i \in S} \sum_{m \in F} d_{im} x_{im} \tag{2.1}$$

This cost is measured in terms of the network delay or number of hops between a switch and its master controller. Hence, the cost decreases if controllers are located close to the switches that are under its control.

2) **Flow-setup cost** (C_p) is the total cost incurred for setting up the flow rules across end-to-end paths. As explained in Fig. 2.12, this cost can be divided into three components. First, the initial path setup request for the flows originated at the switches:

$$C_p{}^R = \sum_{i \in S} \sum_{m \in F} \tau_{ii} x_{im} d_{im} \tag{2.2}$$

Secondly, the intermediate path setup requests at each switch for the flows coming from a neighbor switch controlled by a different controller:

$$C_p{}^Q = \sum_{i \in S} \sum_{j \in S} \sum_{m \in F} \sum_{n \in F} \tau_{ji} x_{jn} (1 - x_{in}) x_{im} d_{im} \tag{2.3}$$

Finally, the rule installation cost incurred for the rule installation messages from the controllers is given by:

$$C_p{}^L = \sum_{i \in S} \sum_{j \in S} \sum_{m \in F} \tau_{ji} x_{im} d_{im} \tag{2.4}$$

Combining Equations (2.2), (2.3), and (2.4), we can derive the flow-setup cost as follows:

$$C_p = C_p{}^R + C_p{}^Q + C_p{}^L \tag{2.5}$$

3) **Synchronization cost** (C_s) represents the number of messages exchanged between controllers in order to maintain a consistent network-wide view in all of them. We assume messages are exchanged every T_x seconds (note that $T_x < T_a$). We also consider critical events that force a controller to instantaneously synchronize state with other controllers. Assuming e is a random variable that represents the occurrence frequency of critical events in the system. We can define the number of inter-controller state synchronization messages generated within time T_a considering both periodic and critical events as follows:

$$N_{\mathcal{E}} = \left\lfloor \frac{T_a}{T_x} \right\rfloor + \int_0^{T_a} e \cdot p(e) de \tag{2.6}$$

Here, $p(e)$ is the probability distribution function of e. The synchronization cost can be defined as follows:

$$C_s = \frac{N_{\mathcal{E}}}{T_a} \sum_{m \in F} \sum_{n \in F} y_m y_n d_{mn} \tag{2.7}$$

This cost is measured in terms of the network delay or hop count between the controllers. While the previous two costs consider the distance between the switch and its master controller, this cost considers the distance between controllers. It decreases with the number of active controllers and the distances between them.

4) **Switch reassignment cost** (C_r) is the cost of assigning a switch to a new controller. Ideally, it is better to avoid frequent reassignment of switches. Assume that the previous assignment is given by the matrix $\tilde{\mathcal{X}} = [\tilde{x}_{im}]_{|S| \times |F|}$. We define the matrix $\mathcal{Z} = [z_{im}]_{|S| \times |F|}$ as the XOR between the new assignment $\mathcal{X}$ and the previous assignment $\tilde{\mathcal{X}}$. In particular,

$z_{im} = 1$ if the assignment of switch i has been changed to (or from) controller m, otherwise $z_{im} = 0$.

$$C_r = \sum_{i \in S} \sum_{m \in F} d_{im} z_{im} \tag{2.8}$$

The objective of our optimization problem is to minimize the weighted sum of the aforementioned four costs and can be expressed as follows:

$$\alpha C_l + \beta C_p + \gamma C_s + \lambda C_r \tag{2.9}$$

Here, α, β, γ, and λ are constants the network operator can use to adjust the relative significance of the four cost components. Furthermore, the following constraints must be satisfied in order to guarantee a feasible solution:

$$\forall_{i \in S}: \sum_{m \in F} x_{im} = 1 \tag{2.10}$$

$$\forall_{m \in F}: \sum_{i \in S} x_{im} \tau_{ii} + \sum_{i \in S} \sum_{j \in S} \sum_{n \in F} \tau_{ji} x_{jn} (1 - x_{in}) x_{im} \leq y_m u_m \tag{2.11}$$

$$\forall_{i \in S, m \in F}: x_{im} d_{im} \leq \delta \tag{2.12}$$

$$\forall_{i \in S, m \in F}: x_{im} \leq y_m \tag{2.13}$$

$$\begin{aligned} \forall_{i \in S, m \in F}: z_{im} &\leq x_{im} + \tilde{x}_{im} \\ z_{im} &\geq x_{im} - \tilde{x}_{im} \\ z_{im} &\geq -x_{im} + \tilde{x}_{im} \\ z_{im} &\leq 2 - x_{im} - \tilde{x}_{im} \end{aligned} \tag{2.14}$$

$$\forall_{i \in S, m \in F}: x_{im}, z_{im} \in \{0, 1\} \tag{2.15}$$

$$\forall_{m \in F}: y_m \in \{0, 1\} \tag{2.16}$$

Constraint (2.10) guarantees that every switch is controlled by exactly one controller at a given time. Inequality (2.11) ensures that a controller can satisfy the path setup requests from the switches assigned to it. Note that the total number of path setup requests to a

controller is composed of all initial and intermediate path setup requests from all switches that it is currently controlling. Inequality (2.12) gives an upper bound δ on the maximum delay between a switch and its designated controller. The condition on assigning a switch to an active controller is represented by Inequality (2.13). The inequalities of (2.14) ensure that z_{im} is the XOR of the variables x_{im} and $\tilde{x}_{im}$. Equations (2.15) and (2.16) indicate that x_{im}, y_m, and z_{im} are binary variables. This formulation generalizes the *Single Source Unsplittable Flow Problem* [90], which is known to be $\mathcal{NP}$-Hard. Therefore, we propose two heuristics to solve this problem that are described in the subsequent sections.

2.6 Proposed Heuristics

In this section, we describe two heuristics for solving DCPP: (i) *DCP-GK*: a greedy approach based on the knapsack problem, and (ii) *DCP-SA*: a simulated annealing based meta-heuristic approach. The input to both heuristics include network topology G, traffic matrix $\mathcal{T}$, previous switch–to–controller assignment $\tilde{\mathcal{X}}$, set of switches S, possible controller locations F, controller capacity vector U, and delay constraint δ. The goal of these heuristics is to find a feasible switch–to–controller assignment that minimizes the cost function expressed in Equation (2.9) based on current network conditions.

2.6.1 Dynamic Controller Provisioning with Greedy Knapsack (DCP-GK)

Here, we model each controller as a knapsack. The capacity of each knapsack is equal to the processing capacity (measured in number of flow-setup requests it can handle per time interval, 60 minutes in our simulations) of its corresponding controller. We consider the switch as the objects to be added in the knapsack. We model the weight of a switch as the number of new flows it generates within the previous time interval, and the profit of taking a switch is the inverse path cost between the switch and that controller. Each iteration of our algorithm activates a single controller. This controller is chosen such that the sum of path costs from that controller to the unassigned switches is minimum and within the given delay bound δ. Then we run the greedy knapsack algorithm to assign switches to that controller. If no switch could be assigned to a controller, it is deactivated. The iterations stop when all the switches are assigned to a controller or no more controllers can be activated. If there are unassigned switches after all the iterations are completed, the switches are assigned randomly between the activated controllers. This approach may

break the capacity and delay constraints. However, this exceptional case occurred very rarely during our simulations.

2.6.2 Dynamic Controller Provisioning with Simulated Annealing (DCP-SA)

The DCP-SA heuristic provides a feasible switch-to-controller assignment $\mathcal{X}$ considering the previous assignment matrix $\tilde{\mathcal{X}}$ as an initial state for simulated annealing. However, due to a change in traffic pattern, $\tilde{\mathcal{X}}$ may violate the capacity constraint depicted in Equation (2.11). Therefore, the objective of Algorithm 1 is to generate a feasible switch–to–controller assignment from the current unfeasible assignment. The output of this algorithm is provided as an input to Algorithm 2 which runs the simulated annealing algorithm to improve the switch–to–controller assignment.

More specifically, Algorithm 1 first identifies the set of controllers F_v for which capacity constraints are violated (line 1). Then in the while loop between line 2 and 16, it tries to lower the load on each $f \in F_v$ by reassigning one or more switches to other controllers without violating the capacity constraint. In line 3 and 4, f is the currently selected controller and S_f is the set of switches assigned to it. In the second nested while loop between line 5 and 14, Algorithm 1 first sorts all switches S_f assigned to controller f according to their rank defined by the following equation:

$$r_i = \sum_{j \in S} \tau_{ij} \tag{2.17}$$

Next, in line 7, let s^* denote the switch with the highest rank in S_f. A set of feasible controllers F_{s^*} is identified for s^* in line 8 such that each controller in F_{s^*} is within the bound δ from s^* and also has sufficient capacity to handle requests from s^*. The algorithm then selects the controller with the smallest remaining capacity $\tilde{f}$ in line 10, and assigns s^* to $\tilde{f}$ (line 11). The intuition is to minimize the fragmentation of remaining capacity of the controllers during the reallocation. The assignment matrix $\tilde{\mathcal{X}}$ is also updated in line 11 by changing the switch's mapping from f to $\tilde{f}$. This reallocation procedure for controller f continues until the capacity constraint for f is satisfied, *i.e.*, aggregated demand of the switches assigned to it becomes less than or equal to its capacity. Algorithm 1 repeats this

Algorithm 1 Algorithm for generating feasible initial state

Require: Topology, G
Traffic Matrix, $\mathcal{T}$
Previous Assignment, $\tilde{\mathcal{X}}$
Set of switches, S
Set of controllers, F
Controller capacity vector, U

Ensure: New Feasible Assignment, $\tilde{\mathcal{X}}$

$F_v \leftarrow$ Set of controllers for which $\tilde{\mathcal{X}}$ violates capacity constraints
while $F_v \neq \emptyset$ **do**
  Select a controller f from F_v
  $S_f \leftarrow$ Set of switches assigned to controller f
  while Capacity of f is violated **do**
    Sort S_f according r_i defined by Equation (2.17)
    $s^* \leftarrow$ first node in S_f
    $F_{s^*} \leftarrow$ Feasible controllers of s^* with remaining capacity greater than the demand of s^*
    if $F_{s^*} \neq \emptyset$ **then**
      $\tilde{f} \leftarrow$ Controller with smallest remaining capacity in F_{s^*}
      Assign s^* from f to $\tilde{f}$ and update $\tilde{\mathcal{X}}$
    end if
    $S_f \leftarrow S_f \setminus \{s^*\}$
  end while
  $F_v \leftarrow F_v \setminus \{f\}$
end while

reallocation procedure for each of the controllers in set F_v until the switch to controller assignment $\tilde{\mathcal{X}}$ becomes feasible.

Due to the change in the traffic pattern in the network, the previous switch to controller assignment, $\tilde{\mathcal{X}}$, may violate the controller capacity constraint depicted in equation 2.11. Therefore, before invocation of the reassignment algorithm (Algorithm 2), first $\tilde{\mathcal{X}}$ is made feasible using Algorithm 1.

Starting from a feasible assignment $\tilde{\mathcal{X}}$, Algorithm 2 uses a variant of simulated annealing to optimize the assignment further. We define the following local search moves for this algorithm:

- **Relocate Switch**: selects a switch randomly and assigns it to a different active controller. If no switch is assigned to a controller after this move, it is deactivated.
- **Swap switches**: selects two switches randomly from two different controllers and swap their assignments.
- **Activate controller**: activates a randomly chosen inactive controller.
- **Merge assignments**: randomly selects two controllers and reassigns all switches of one controller to the other. The idle controller is then deactivated.

Algorithm 2 denotes the implementation of simulated annealing for switch reassignment. To accelerate future computations, Algorithm 2 pre-calculates the feasible controller set F_S for all switches S in line 3 such that delay constraint (equation 2.12) is satisfied. Next, the for loop starting at line 5 maintains the temperature based annealing process. The temperature is adjusted using the *Schedule(t)* procedure (line 6) in each iteration of the for loop. Then, the loop between line 10 and 23, uses the *Successor* procedure to return a random next state from the current state using one of the aforementioned moves. This procedure always returns a feasible successor such that no constraint is violated. Following the standard simulated annealing procedure, a move is always taken if it reduces the cost according to Equation 2.9. However, to overcome the local minima, a move that leads to a worse state can also be taken with probability $e^{\frac{\Delta}{Temp}}$, where $Temp$ and Δ are the current temperature and difference in cost between current and next states, respectively. The temperature is initialized so that Algorithm 2 can take many worse moves at the start to explore the search space. However, it is decremented by *Schedule* function in such a way that the probability of taking worse moves diminishes with time. For each temperature, state space is explored for a pre-defined number of moves. At each step, the algorithm stores the best solution in $\mathcal{X}$. Finally, the algorithm will stop and return the assignment $\mathcal{X}$ with the lowest cost seen so far when the temperature falls to zero.

Algorithm 2 Reassignment algorithm

Require: Topology, G
Traffic Matrix, $\mathcal{T}$
Feasible Previous Assignment, $\tilde{\mathcal{X}}$
Set of switches, S
Set of controllers, F
Controller capacity vector, U

Ensure: New Assignment, $\mathcal{X}$

$\mathcal{X} \leftarrow \tilde{\mathcal{X}}$
$F_S \leftarrow$ Feasible controllers for S considering delay constraints
Select an initial temperature $Temp > 0$
$current \leftarrow \tilde{\mathcal{X}}$
for $t \leftarrow 1$ to ∞ **do**
  $Temp \leftarrow$**Schedule**(t)
  if $Temp = 0$ **then**
    break
  end if
  $i \leftarrow 1$
  repeat
    $next \leftarrow$ **Successor**$(current, F_S)$
    $\Delta \leftarrow$ **Cost**$(current) -$ **Cost**$(next)$
    if $\Delta > 0$ **then**
      $current \leftarrow next$
    else
      $current \leftarrow next$ only with $e^{\frac{\Delta}{Temp}}$ probability
    end if
    if Cost$(current) <$ **Cost**$(\mathcal{X})$ **then**
      $\mathcal{X} \leftarrow current$
    end if
    $i \leftarrow i + 1$
  until $i \neq N$
end for

2.7 Evaluation

We evaluate the performance of our proposed algorithms through extensive simulations. We opted for an in-house simulator where we simulate the propagation delays between switch-to-switch, switch-to-host, and switch-to-controller. Controller capacity is simulated using the results provided by Tootoonchian *et al.* [28] for the NOX [67] controller. All our simulations are conducted on a machine with dual quad-core 2.4GHz Intel Xeon E5620 processors and 12-GB of RAM. In the following, we first describe in detail the simulation setup and the dataset we used. We then describe the metrics used to evaluate the effectiveness of our proposed system. Finally, we compare our DCP algorithms (DCP-GK and DCP-SA) with two static scenarios: in the first case, a single controller is used for the entire network (1-CRTL), while in the second, each switch is provided its own controller (N-CTRL), *i.e.*, a network with N switches will have N controllers.

2.7.1 Simulation Setup

In our experiments, we simulate two different ISP topologies RF-I and RF-II with inter-node latencies obtained from the RocketFuel repository [91]. Table 2.1 reports the number of nodes and links in the topologies used in this work. We assume each node in the RocketFuel topology to be an OpenFlow switch, and controllers can be dynamically provisioned at any of these switch locations. Controllers communicate with each other to exchange and synchronize switch and port-level statistics. Each controller computes a path for a new incoming flow from the information it has about the network in its local database and sets up paths according to the method described in Section 2.4.2.

Table 2.1: Characteristics of the simulated ISP topologies

Topology ID	Nodes	Links
RF-I	79	294
RF-II	108	306

We simulate TCP flows between the end hosts, where the end hosts of each flow are chosen randomly. To make the traces more realistic, we generated the flows according to the distribution of flow sizes, flow inter-arrival times, and the number of concurrent flows reported in a recent study on network traffic characterization [92]. The generated traffic spans 48 hours capturing the time-of-day effect.

The simulated annealing based heuristic involves a number of tuning parameters. We performed several test runs to determine a set of suitable but generic values for them, which are reported in Table 2.2. We used the same parameter values for all of our experiments.

Table 2.2: Simulated annealing parameters

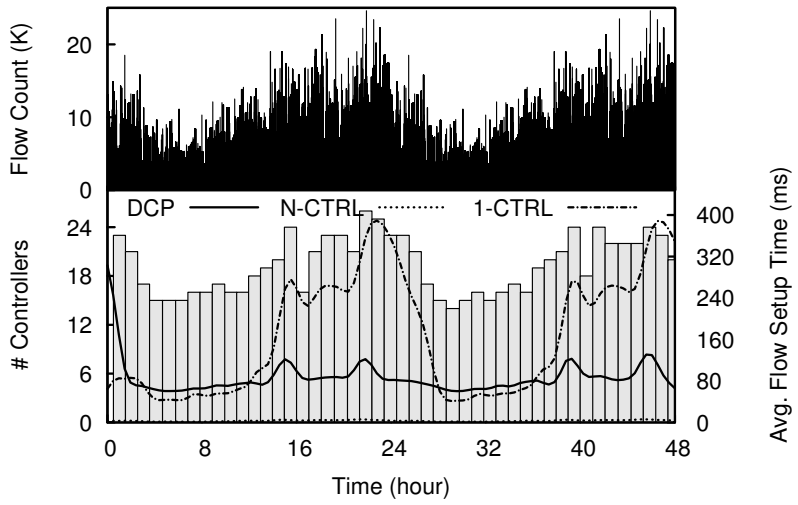
Parameter	Value	Parameter	Value
α	1	δ	40
β	1	Initial Temperature	1000
γ	1000	Temperature decrement rate(r)	0.95
λ	10000	Temperature update function	$T_{i+1} = rT_i$

The management system operates as follows: the monitoring module periodically pulls statistics from the controllers, the reassignment module runs a heuristic (either DCP-GK or DCP-SA) using these statistics to find the next switch-to-controller assignment, and the provisioning module assigns switches to their controllers according to the assignment generated by the reassignment module.

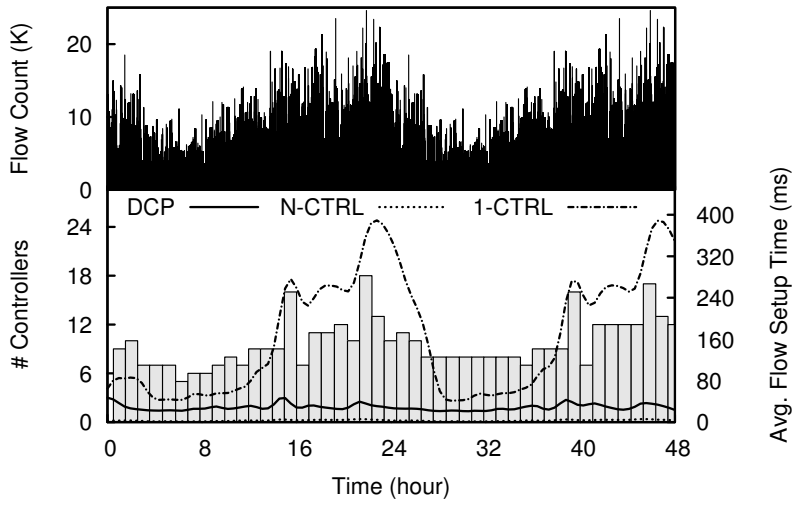
In order to show the effectiveness of our system, we have analyzed three metrics, namely, the number of controllers, flow-setup time, and communication overhead in the number of exchanged network messages. We plot the change in the number of controllers over time to show how it varies with traffic. We measure the flow-setup time at regular time intervals and report that against traffic and number of controllers. We also report the CDF of flow-setup time to show the effectiveness of our system. We also measure the communication cost, which is the total number of messages exchanged between the controllers.

2.7.2 Results

The top part of Figure 2.13 depicts the distribution of traffic used for our evaluation. This traffic is generated using the flow size and flow inter-arrival time distributions provided in [92]. The generated traffic trace spans two days (48 hours) capturing the time-of-day effect. We run each simulation for 48 hours with the reassignment heuristic running every 60 minute. The reassignment interval can be further tuned through a more detailed analysis of the traffic. At each interval, we compute the average flow-setup time, the set of active controllers, and the number of exchanged messages between the active controllers.

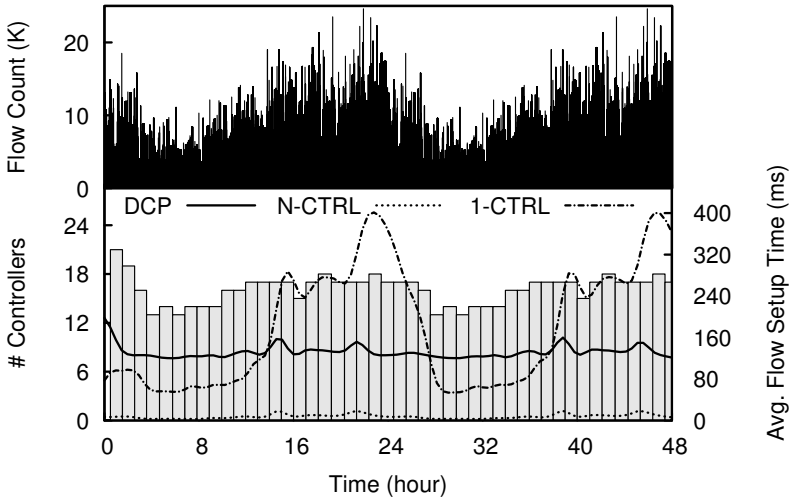


(a) DCP-GK on RF-I

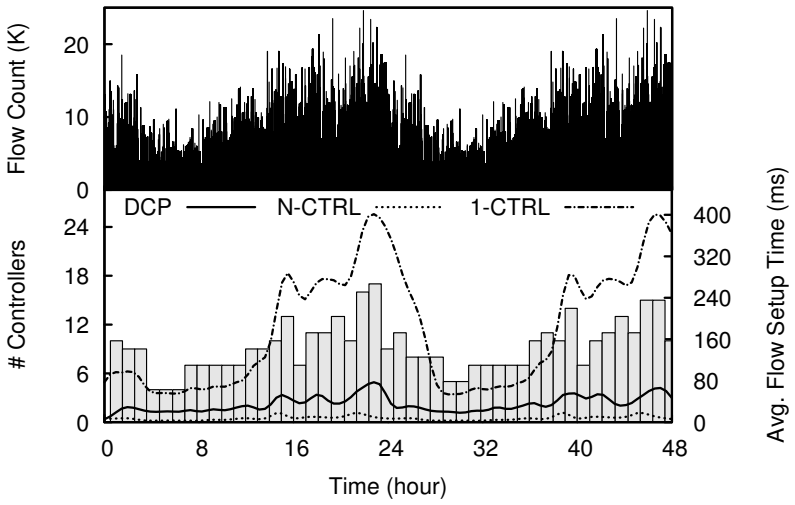


(b) DCP-SA on RF-I

Figure 2.13: RF-I: Controller count and flow-setup time vs. time



(a) DCP-GK on RF-II



(b) DCP-SA on RF-II

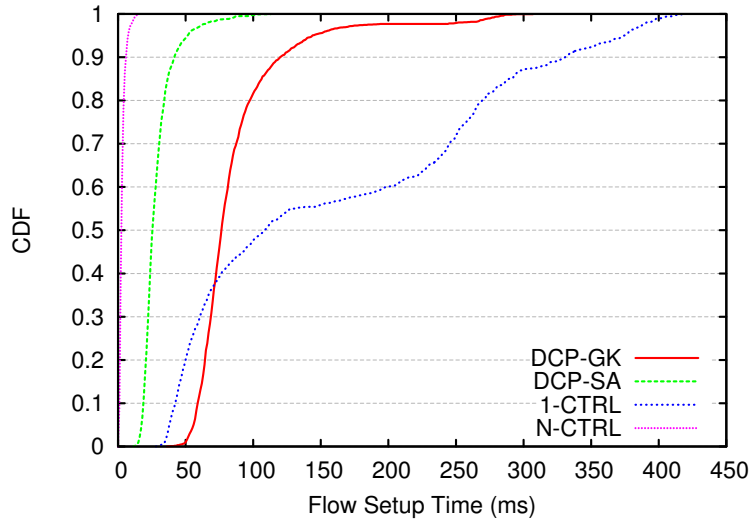
Figure 2.14: RF-II: Controller count and flow-setup time vs. time

Fig. 2.13 and 2.14 show the number of active controllers and average flow-setup time during each interval for both topologies. For the one controller (1-CTRL) case, flow-setup time varies with traffic load. If there is a peak in traffic, then flow-setup time also increases. A single controller cannot keep the flow-setup time consistent or within acceptable limits, which is reported to be 200ms in [93] for mesh restoration. Hence, a single controller cannot provide any service guarantees. On the other hand in the N-CTRL case, the flow-setup time is almost zero as expected. However, in this case, the messaging overhead is much higher (as shown in Fig. 2.16 and explained later in this section).

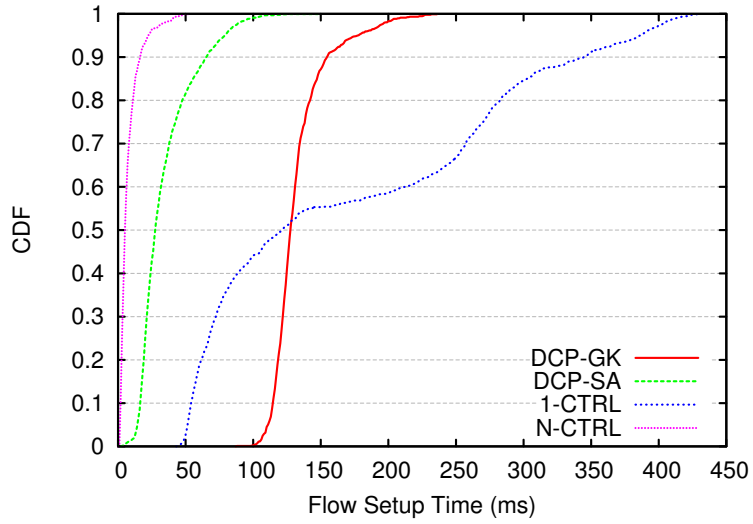
Fig. 2.13(a) and Fig. 2.13(b) report the above mentioned metrics for topology RF-I, using greedy knapsack (DCP-GK) and simulated annealing (DCP-SA) heuristics, respectively. DCP-GK keeps the flow-setup time within 140ms and manages to keep it consistent even during traffic spikes. Even though we can see small spikes in flow-setup time, none of them is as high as for the 1-CTRL case. DCP-GK also uses much fewer controllers than N-CTRL. The maximum number of controllers used by DCP-GK is only 25, during time interval 22, which is around 30% of the N-CTRL (79 controllers) case. From Fig. 2.13(b), we can see that DCP-SA performs much better than DCP-GK and 1-CTRL. Flow-setup time is within 60ms and number of controllers is less than that of both DCP-GK and 1-CTRL. It uses a maximum of 18 controllers that are around only 23% of the N-CTRL case. Effect of traffic spikes is further reduced in this case and the flow-setup time is almost constant throughout 48 hours of simulated time. This shows the effectiveness of our dynamic controller provisioning mechanism. Similar behavior is also observed for RF-II as reported in Fig. 2.14(a) and 2.14(b).

Although DCP-SA outperforms DCP-GK in both the number of controllers and flow-setup time, there is a penalty. DCP-SA requires much longer time to run than DCP-GK. In our simulation setup, DCP-SA took around 2 minutes to perform one reassignment for topology RF-I, whereas DCP-GK took only 0.41 seconds. For topology RF-II, DCP-SA took around 4 minutes, and DCP-GK took only 0.44 seconds. So, DCP-SA is preferable for near-optimal solutions, and DCP-GK is suitable when we need solutions within a small amount of time.

Fig. 2.15(a) and Fig. 2.15(b) show the CDFs of flow-setup time for topologies RF-I and RF-II, respectively. We can see that DCP-SA outperforms both DCP-GK and 1-CTRL in both cases by a large margin. DCP-SA always provides shorter flow-setup time than 1-CTRL and all flows are set up within the acceptable range of 200ms. For RF-I, DCP-SA takes at most 120ms, and for RF-II, it takes at most 150ms. While DCP-GK takes longer, it completes 99% flow-setups within the acceptable range of 200ms for both topologies. On the other hand, 1-CTRL can complete only 60% flow-setups within 200ms and the maximum time it takes is close to 450ms (more than twice of the acceptable range) for



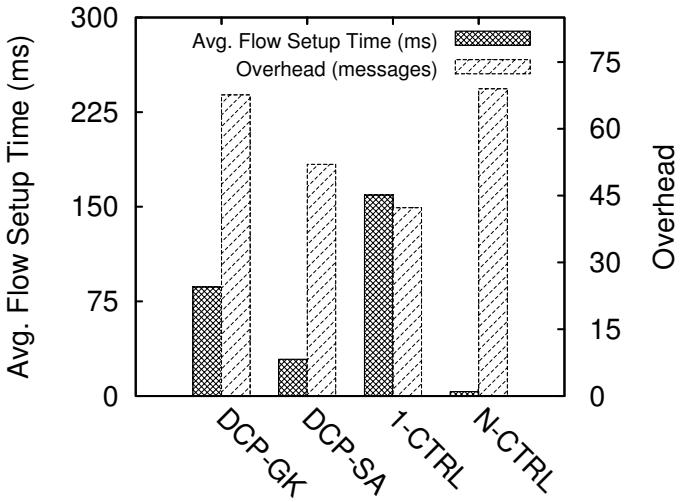
(a) RF-I



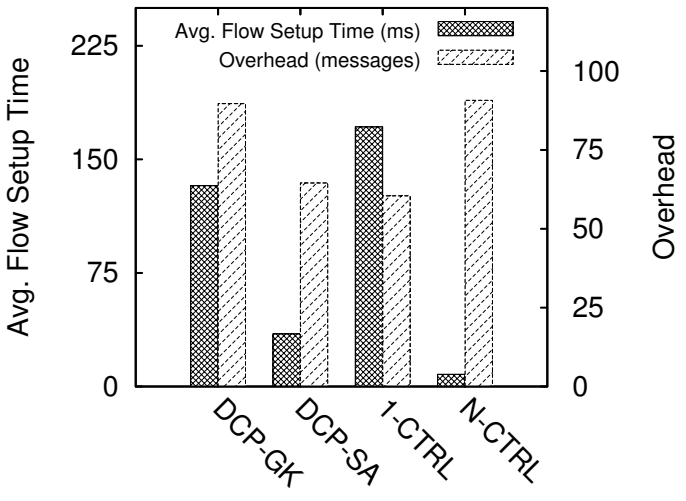
(b) RF-II

Figure 2.15: CDF of flow-setup time

both topologies. DCP-GK cannot outperform 1-CTRL for low flow-setup time, which is evident in Fig. 2.13(a) and Fig. 2.14(a). This happens during low traffic load conditions. This is a direct consequence of the greediness of this heuristic, as it chooses the best



(a) RF-I



(b) RF-II

Figure 2.16: Summary of overhead and average flow-setup time

controller at each stage without looking ahead and thereby missing a better solution. The N-CTRL case shows the lowest flow-setup time, but this is a hypothetical, un-realistic lower bound. Clearly, *connecting one controller per switch* – is not an acceptable solution for this problem. We included it for comparing with the absolute lowest possible flow-setup time.

Fig. 2.16(a) and Fig. 2.16(b) report the messaging overhead and average flow-setup time for both topologies. 1-CTRL has the lowest messaging overhead as there is no synchronization and controller-to-controller communication overhead. On the other hand, N-CTRL has the highest messaging overhead as every controller is communicating with every other controller. Messaging overhead for DCP-GK and DCP-SA is in between these two. Overhead for DCP-SA is smaller than DCP-GK as it uses a fewer number of controllers and is very close to the 1-CTRL (lower bound) case for both topologies. As mentioned earlier, average flow-setup time for DCP-SA is lower than both DCP-GK and 1-CTRL, but higher than N-CTRL. For topology RF-I and RF-II, average flow-setup time for DCP-SA is 29 and 34ms, respectively. For the N-CTRL case, flow-setup times are much lower (3.5 and 9ms, respectively). So, DCP-SA provides flow-setup times very close to N-CTRL (hypothetical lower bound for the flow-setup time) case and also incurs messaging overhead very close to 1-CTRL (lower bound for messaging overhead) case. On the other hand, DCP-GK does not provide as good result as DCP-SA, but the solutions are quite good and require fractions of seconds to run.

To summarize, running a single controller causes high flow-setup time, as propagation delay between controller and switches are higher, and flow-setup requests can get queued at the controller because of limited processing capacity. On the other hand, running one controller per switch can provide close to zero flow-setup times, but incurs much higher communication overhead. Our system achieves a balance between flow-setup time and messaging overhead. Our simulation results show that DCP-SA and DCP-GK succeed to find a right trade-off between these two extremes and provide near-optimal solutions.

2.8 Conclusion

In this work, we identified and formulated the Dynamic Controller Provisioning Problem (DCPP) in SDN. We proposed a management system for dynamically deploying multiple controllers. We also provided a mathematical formulation of DCPP as an ILP. Since DCPP is an $\mathcal{NP}$-hard problem, we provided two heuristic algorithms (DCP-GK and DCP-SA) to solve it. The evaluation results presented in this work provide important insights on various controller provisioning strategies. Running a single controller causes high flow-setup delay,

as propagation delay between controller and switches are higher and flow-setup requests can get queued at the controller because of limited processing capacity. On the other hand, running one controller per switch can provide close to zero flow-setup times, but incurs significant overhead for inter-controller communication. Our simulation results show that our solution can achieve lower flow-setup time and minimal communication overhead compared to that of the static version of the problem (using a single or multiple controllers). Our system achieves a balance between flow-setup time and messaging overhead. Evaluation results show that DCP-SA and DCP-GK succeed to find a right trade-off between these two extremes and provide near-optimal solutions. DCP-SA provides better results than DCP-GK, but takes longer to converge.

Chapter 3

Orchestrating Virtual Network Functions

3.1 Introduction

Today's communication networks ubiquitously deploy hardware middleboxes or network appliances to offer different types of network services. Examples of such middleboxes include firewalls, proxies, WAN optimizers, Intrusion Detection Systems (IDSs), and Intrusion Prevention Systems (IPSs). Middleboxes are used to achieve various kinds of performance and security related objectives for a network [13, 14]. Recent studies show that the number of deployed middleboxes is very close to the number of routers in enterprise and data center networks [14, 33]. Even though middleboxes have become an integral part of modern networks, they come with high Capital Expenditure (CAPEX) and Operational Expenditure (OPEX). They are usually vendor specific, vertically-integrated, expensive, and require specially trained personnel for deployment and maintenance. Moreover, it is often impossible to add new functionality to an existing middlebox, which makes it very difficult and cumbersome for the network operator to deploy new services or technologies. In many cases, the operator is compelled to purchase new hardware, which may lead to an assessment of new equipment, network deployment strategy, or even considering new vendors who can offer better products. Reassessment of business and operational strategies increases cost and time required to introduce new services or technologies that may lead to loss of customers and revenue.

Another set of problems arise from the fact that most often network traffic is required to pass through multiple stages of middlebox processing in a particular order, *e.g.*, a traffic

flow may be required to go through a firewall, then an IDS, and finally through a proxy [37]. This phenomenon is very common for middleboxes and is typically referred to as Service Function Chaining (SFC) [38]. The IETF Network and Service Chaining Working Group has several IETF drafts that demonstrate middlebox chaining use-cases in operator [94], mobile [40], and data center networks [95]. The task of sequencing in-network middlebox processing is commonly referred to as *middlebox orchestration*. Currently, middleboxes are attached to fixed locations within a network. Traffic flows are routed through the required sequence of middleboxes by manually crafting the routing table entries in the routers, which is a cumbersome and error-prone process. Moreover, the fixed location of middleboxes restricts the traffic routing paths from efficiently utilizing the available transport capacity of the network, as the attachment points of middleboxes become hot-spots within the network.

An emerging and promising technology that can address these limitations is Network Function Virtualization (NFV) [44, 11]. It proposes to move packet processing from hardware middleboxes to software middleboxes or Virtual Network Functions (VNFs) running on commodity (*e.g.*, `x86` based systems) servers. This approach will not hamper performance as many state-of-the-art software middleboxes have already shown the potential to achieve near-hardware performance [45, 46]. NFV provides ample opportunities for network optimization and cost reduction. Previously, middleboxes were hardware appliances placed at fixed locations, but now a VNF can be deployed on any server in the network. VNF locations can be determined intelligently to ensure efficient traffic routing. NFV opens-up the opportunity to simultaneously optimize VNF locations and traffic routing paths; thereby reducing network OPEX by maintaining a minimal set of active servers and switches to lower the overall energy consumption of the network.

VNF chains can be orchestrated by dynamically deploying a composition of VNFs either on a single server or on a cluster of servers. However, several issues need to be considered before provisioning VNFs: (i) the cost of deploying a new VNF, (ii) energy cost for running a VNF, (iii) the cost of forwarding traffic to and from a VNF, and (iv) fragmentation of the underlying physical resource pool. Placing just enough VNFs to match traffic processing requirements may yield the lowest deployment and energy cost, but steering traffic through these VNFs will increase traffic forwarding cost and may eventually lead to Service Level Objective (SLO) violations. On the other hand, one may try to always forward traffic through the shortest possible path by deploying VNFs in all possible locations. This approach may avoid SLO violations, but will surely lead to huge deployment and energy costs. An optimal VNF orchestration strategy must address these issues during the cost minimization process. Moreover, it must avoid SLO violations and satisfy the capacity constraints of the physical servers and physical links. We refer to this problem as the

Virtualized Network Function Orchestration Problem (VNF-OP).

VNF-OP has similarities with generic Virtual Machine (VM) placement [96] or Constrained Shortest Path (CSP) [97] problems. However, VM placement problems only consider the resource capacities of physical servers and links [98, 99], but in case of VNF-OP, the number of required VNFs is not known in advance and we need to consider the traffic processing capacity of a VNF as well. CSP problems do not consider finding a shortest path through an ordered sequence of nodes. To the best of our knowledge, there has been no prior work that explores the benefits of dynamic VNF orchestration, nor solves the related algorithmic problems. This work provides a mathematical formulation for VNF-OP along with both optimal and heuristic solutions to solve this problem.

This work makes the following contributions:

- We provide the first quantifiable results showing that dynamic VNF orchestration can have more than 4 times reduction in OPEX compared to hardware middleboxes.
- The problem is formulated as an Integer Linear Program (ILP) and implemented in CPLEX[1] to find optimal solutions for small scale networks.
- We prove the NP-hardness of VNF-OP by a reduction from the Capacitated Plant Location Problem with Single Source constraints.
- We propose a fast heuristic algorithm that can find near-optimal solutions for large scale networks within seconds.
- The heuristic's performance in terms of solution quality and scalability is evaluated using both real-world and synthetic topologies and traffic traces.
- Finally, we compared the performance of our heuristic with related work from the literature [100].

The rest of the chapter is organized as follows: Section 3.2 provides background on network function virtualization, then we explain the mathematical model used for our system and formally define the VNF Orchestration Problem (Section 3.3). Then the problem formulation is presented (Section 3.4). Next, a heuristic is proposed to obtain near-optimal solutions (Section 3.5). We validate our solution through trace driven simulations on real-world network topologies (Section 3.6) and finally, we summarize the results with some future research directions (Section 3.8).

[1]http://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/

3.2 Network Function Virtualization

The primary objective of NFV is to revolutionize the service provisioning architecture of communication networks. The transport capacity and speed of current networks have continued to improve; however, they still struggle to handle the changing landscape of requirements imposed by different applications and services. As discusses above, traditional network services are materialized by manually configuring routing table entries to guide traffic through a sequence of middleboxes. Example of middleboxes include firewalls, proxies, intrusion detection and prevention systems, load-balancers, and video optimizers. Each of these middleboxes provides a particular set of packet processing capabilities, *e.g.*, a firewall forwards or drops packets based on a pre-configured set of rules, a load-balancer distributes traffic among a set of servers, and an intrusion detection system identifies malicious traffic entering the network. The processing logic offered by a middlebox is called a network function, and the software entity that implements this function in a virtualized environment is called a Virtual Network Function or VNF. Similar to the decoupling of data and control planes in SDN, NFV proposes to decouple the network functions from the underlying hardware.

The first and foremost benefit of NFV is vendor-independence. By decoupling the network functions from the underlying hardware, it enables network operators to utilize standardized and commodity off-the-shelf hardware. The software running on hardware middleboxes are tightly integrated with the hardware circuits and focuses exclusively on performing a specific set of functions. The device is closed-source, as a network operator does not have access to the embedded software. In contrast, in NFV, the software implementing a network function can be procured from the hardware vendor, third-party companies, or developed in-house. This decoupling lowers the barrier to market-entry for new companies; creating a competitive market-place that reduces the initial and maintenance cost of VNF software. The ability to utilize commodity hardware and the evolution towards an open VNF market will significantly reduce a network operator's CAPEX. It is otherwise hard for the network operator to break-free from a vendor as it would require significant capital spending and management overhead.

NFV has the potential to change all that by decoupling the packet processing tasks from the underlying hardware through virtualization. Moreover, NFV provides flexibility, agility, and better programmability for service composition, and reduction in time for new service deployment. Virtual network functions can be managed autonomously similar to the way applications are managed in a cloud data center. NFV also opens-up the network infrastructure to a broader range of customers. Third-party content or service providers can rent resources from network operators which are typically much closer to the end-user

and provide innovative services and generate new revenue streams that are not possible with the current proprietary vendor-locked architecture.

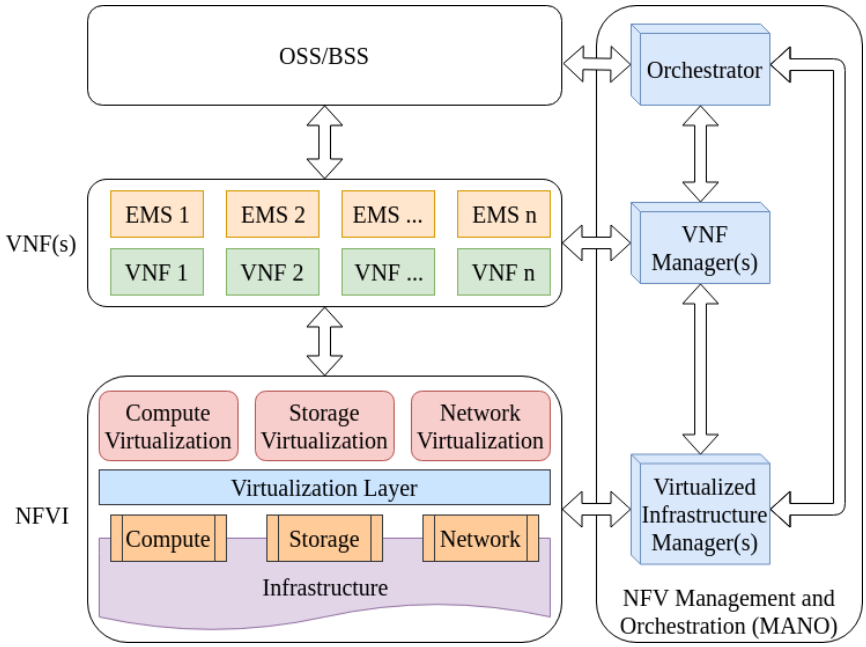


Figure 3.1: NFV architectural framework

The European Telecommunications Standards Institute (ETSI) [101] formed the NFV Industry Specification Group (NFV ISG) to drive research and development in this area. The NFV ISG proposed an architectural framework for NFV, which is shown in Figure 3.1. The NFV Infrastructure (NFVI) consists of the underlying hardware infrastructure including compute, storage, and network resources. A Virtualized Infrastructure Manager (VIM) (*e.g.*, OpenStack [7]) manages NFVI and utilizes existing virtualization technologies to provide virtual resources for VNFs. The VNFs represent virtualized instances of different network functions. Each VNF has a corresponding Element Management System (EMS) that provides management and control functionality for that VNF. The VNF Manager provides life-cycle management functionality for the software components of a VNF. The Orchestrator manages the life-cycle of VNFs and SFCs though the VIM and VNF managers. It utilizes resource allocation and placement algorithms to ensure optimal usage of both physical and software resources. The VIM, VNF manager, and orchestrator constitute the NFV Management and Orchestration (MANO) system. MANO interacts with the Operational and Business Support Systems (OSS/BSS) of the operator to manage the operational and business aspects of the network.

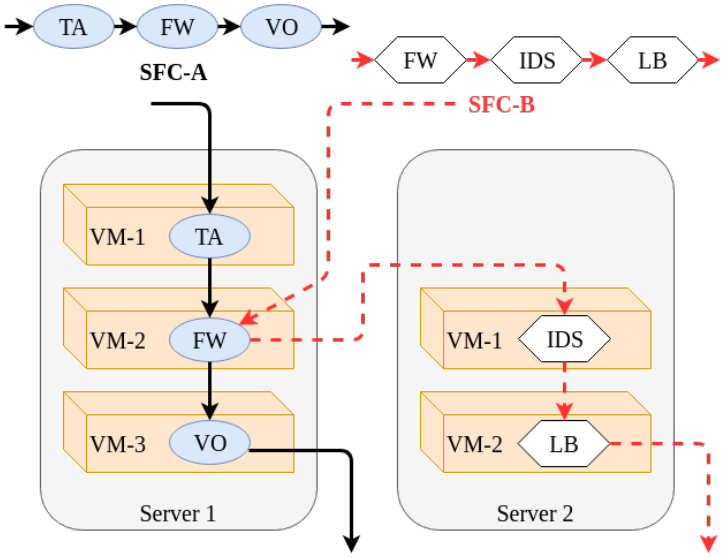


Figure 3.2: VM sharing and embedding of SFC-A and SFC-B

A VNF can be deployed on servers, VMs or containers. In this work, we assume that VNFs are deployed within VMs, and each VNF can support multiple traffic flows. Figure 3.2 shows an example embedding of two SFCs: SFC-A and SFC-B. SFC-A consists of the chain traffic analyzer, firewall, and video optimizer (`TA` $\rightarrow$ `FW` $\rightarrow$ `VO`), and SFC-B consists of firewall, intrusion detection system, and load-balancer (`FW` $\rightarrow$ `IDS` $\rightarrow$ `LB`). The chains are embedded on two servers. Server-1 and server-2 are hosting three and two VMs, respectively. The VNF for firewall, hosted on VM-2 of server-1, is shared by the two chains. In VNF-OP we need to determine the placement and traffic routing paths for SFCs in such a manner that the inter-SFC VNF sharing can be maximized without violating SLOs. Hence, our focus in this work can be summarized as devising algorithms for efficient SFC and VNF placement while ensuring SLOs and reducing OPEX through optimal energy utilization.

3.3 Mathematical Model and Problem Definition

In this section we introduce the mathematical model for our system and formally define the VNF Orchestration Problem.

3.3.1 Physical Network

We represent the physical network as an undirected graph $\bar{G} = (\bar{S}, \bar{L})$, where $\bar{S}$ and $\bar{L}$ denote the set of switches and links, respectively. We assume that VNFs can be deployed on commodity servers located within the network. These network locations are traditionally known as Point-of-Presences or PoPs. The set $\bar{N}$ represents these servers and the binary variable $\bar{h}_{\bar{n}\bar{s}} \in \{0, 1\}$ indicates whether server $\bar{n} \in \bar{N}$ is attached to switch $\bar{s} \in \bar{S}$.

$$\bar{h}_{\bar{n}\bar{s}} = \begin{cases} 1 & \text{if server } \bar{n} \in \bar{N} \text{ is attached to switch } \bar{s} \in \bar{S}, \\ 0 & \text{otherwise.} \end{cases}$$

Let, R denote the set of resources (CPU, memory, and disk) offered by each server. The resource capacity of server $\bar{n}$ is denoted by $c^r_{\bar{n}} \in \mathbb{R}^+, \ \forall\, r \in R$. The bandwidth capacity and propagation delay of a physical link $(\bar{u}, \bar{v}) \in \bar{L}$ are represented by $\beta_{\bar{u}\bar{v}} \in \mathbb{R}^+$ and $\delta_{\bar{u}\bar{v}} \in \mathbb{R}^+$, respectively. We also define $\eta(\bar{u})$ as the set of neighbors for switch $\bar{u}$.

$$\eta(\bar{u}) = \{\bar{v} \mid (\bar{u}, \bar{v}) \in \bar{L} \text{ or } (\bar{v}, \bar{u}) \in \bar{L}\}, \ \bar{u}, \bar{v} \in \bar{S}$$

3.3.2 Virtualized Network Functions (VNFs)

Different types of VNFs (*e.g.*, firewall, IDS, IPS, and proxy) can be provisioned in a network. The possible VNF types are represented by the set P. Each VNF type p has a specific deployment cost, resource requirements, processing capacity, and processing delay represented by $\mathcal{D}^+_p$, $\kappa^r_p \in \mathbb{R}^+ (\forall r \in R)$, c_p (in Mbps), and δ_p (in ms), respectively. These quantities are explained below:

- **Deployment Cost ($\mathcal{D}^+_p$)** includes the cost of image transfer and booting a VNF of type p on a server.
- **Resource Requirement (κ^r_p)** is the amount of resource of category r that must be allocated to a type p VNF.
- **Processing Capacity (c_p)** represents the amount of traffic (in Mbps) a type p VNF can process.
- **Processing Delay (δ_p)** is the average delay (in ms) experienced by a packet when traversing through a VNF of type p.

The actual values of the above mentioned quantities are highly implementation specific and depend on a lot of factors. Here, we have assumed approximate values for these quantities to simplify the mathematical model.

There can be certain hardware requirements (*e.g.*, hardware-accelerated encryption for Deep Packet Inspection (DPI)) that may prevent a server from running a particular type of VNF. Furthermore, the network manager may have preferences regarding provisioning a particular type of VNF on a particular set of servers, *e.g.*, Firewalls should be deployed close to the network edge. So, we assume that for each VNF type there is a set of servers on which it can be provisioned. The following binary variable represents this placement constraint:

$$d_{\bar{n}p} = \begin{cases} 1 & \text{if VNF type } p \in P \text{ can be provisioned on } \bar{n}, \\ 0 & \text{otherwise.} \end{cases}$$

3.3.3 Traffic Request

We assume that the network operator is receiving requests for setting up paths for different kinds of traffic (*e.g.*, VPN setup, security features, and new application or service in a data center). A traffic request is represented by a 6-tuple $t = \langle \bar{u}^t, \bar{v}^t, \Psi^t, \beta^t, \delta^t, \omega^t \rangle$, where $\bar{u}^t, \bar{v}^t \in \bar{S}$ denote the ingress and egress switches, respectively. $\beta^t \in \mathbb{R}^+$ is the bandwidth demand of the traffic. δ^t is the maximum allowed propagation delay according to Service Level Agreement (SLA). Ψ^t represents the ordered VNF sequence the traffic must pass through (*e.g.*, Firewall $\rightarrowtail$ IDS $\rightarrowtail$ Proxy). l_{Ψ^t} denotes the length of Ψ^t and ω^t denotes the policy to determine SLO violation penalties.

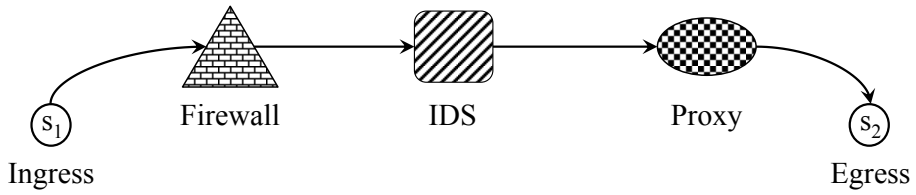


Figure 3.3: VNF chain

In our mathematical model, we transform a VNF sequence Ψ^t into a directed acyclic graph $G^t = (N^t, L^t)$, where N^t represents the set of traffic nodes (VNFs, ingress and egress switches) and L^t denotes the links between them. Figure 3.3 shows a sample VNF chains.

Here, traffic flows through the chain Firewall $\rightarrowtail$ IDS $\rightarrowtail$ Proxy. Modeling the traffic flow in this way makes it easy for the provisioning process to ensure that it passes though the correct sequence of VNFs. We also define $\eta^t(n_1)$ to represent the neighbors of $n_1 \in N^t$:

$$\eta^t(n_1) = \{n_2 \mid (n_1, n_2) \in L^t\},\ n_1, n_2 \in N^t$$

Without loss of generality, we consider $n_2 > n_1$ $(n_1, n_2 \in N^t)$, *iff* n_2 appears after n_1 in the topological order of G^t.

Next, we define a binary variable $g^t_{np} \in \{0, 1\}$ to indicate the type of a node $n \in N^t$

$$g^t_{np} = \begin{cases} 1 & \text{if node } n \in N^t \text{ is of type } p \in P, \\ 0 & \text{otherwise.} \end{cases}$$

3.3.4 VNF Orchestration Problem (VNF-OP)

We consider a scenario where an operational network is serving a set of traffic $\hat{T}$. It has a set of VNFs already deployed and the routing paths for the traffic in $\hat{T}$ are also provisioned. Now, the network operator is receiving new traffic requests and wants to provision the required VNFs and routing paths for them. The network operator can choose to provision resources for one traffic request at a time or leverage a lookahead interval by accumulating a number of traffic requests and provision resources in batches. Determining the optimal number or volume of traffic or the length of the lookahead interval for each batch is an interesting research challenge that is beyond the scope of this work and we plan to pursue it in the future. In the rest of this chapter, we denote a new traffic batch by T. Based on the operator's choice, a batch may contain just one or multiple traffic requests.

In the VNF-OP, we are given a physical network topology, VNF specifications, current network status and a set of new traffic requests. Our objective is to minimize the overall network OPEX and physical resource fragmentation by (i) provisioning an optimal number of VNFs, (ii) placing them at the optimal locations, and (iii) finding the optimal routing paths for each traffic request, while respecting the capacity constraints (*e.g.*, physical servers, links, and VNFs) and ensuring that traffic passes through the proper VNF sequence.

OPEX: In this work, we consider the network OPEX to be composed of the following four cost components:

- *VNF deployment cost:* we need to complete tasks like transferring a VM image, booting it and attaching it to devices before deploying a VNF. We assign a cost (in dollars) to these operations.

- Energy cost: it represents the cost of energy consumption by the active servers. A server is considered active if it has at least one active VNF. Servers consume power based on the amount of resources (*e.g.*, CPU, memory, and disk) under use. A server is assumed to be in the idle state if it does not have any active VNFs [102].

- Traffic forwarding cost: traffic forwarding cost may be incurred from two sources: (i) leasing cost of transit links [103] and (ii) energy consumption of the network devices (*e.g.*, switches and routers). In the rest of this chapter, we use the terms 'traffic forwarding cost' and 'transit cost' interchangeably.

- Penalty for Service Level Agreement (SLA) violation: this cost component represents the penalty that must be paid to the customer for SLA violations, *e.g.*, if a traffic experienced more that the maximum allowed propagation delay.

Resource Fragmentation: We compute physical resource fragmentation by measuring the percentage of idle resources for the active servers and links. We want to minimize fragmentation as this approach eventually increases the possibility of accommodating more traffic on the same physical resources.

3.4 Problem Formulation and Complexity Analysis

VNF-OP is a considerably harder problem to solve than traditional Virtual Network (VN) embedding problems [104]. There is no node ordering requirement in VN embedding, while in VNF-OP we need to preserve the ordering of VNFs. Moreover, in VNF-OP we need to respect the processing capacity constraints of servers and the VNFs to be deployed. How many VNFs are to be deployed is not known in advance, rather it is an outcome of the optimization process. Multi-dimensional Bin Packing [105] can also be used to solve VNF-OP, but this will result in a *nested* bin packing problem. In the first layer traffic needs to be packed into VNFs and in the next layer VNFs need to be packed into the physical servers. The fact that the number and locations of VNFs is not known in advance, results in quadratic constraints for resource capacity and renders the problem unsolvable even for very small instances by existing optimization solvers. In this work, we address these challenges by intelligently augmenting the physical network, as explained in the following.

3.4.1 Physical Network Transformation

We transform the physical network to generate an augmented pseudo-network that reduces the complexity involved in solving VNF-OP. The transformation process is performed in the following two steps as shown in Figure 3.4:

VNF Enumeration

A part of a physical network topology is shown in Figure 3.4(a). Here, we have three switches ($s1, s2$ and $s3$) and a server $n2$ connected to switch $s2$. We enumerate all possible VNFs in this step by finding the maximum number for each VNF type that can be deployed on each server. We calculate this number based on the resource capacity of the server and the resource requirement of a type of VNF. For example, if a server has 16 cores, and CPU requirement for Firewall and IDS are 4 and 8 cores, respectively, we can deploy 4 Firewalls and 2 IDSs on it. In Figure 3.4(b) we show enumerated VNFs for server $n2$.

We denote the set of these VNFs (called pseudo-VNFs) by $\mathcal{M}$. Each VNF $m \in \mathcal{M}$ is implicitly attached to a server $\bar{n} \in \bar{N}$. We also attach two additional pseudo-VNFs to each server to represent the ingress and egress points of a traffic request. We use the function $\zeta(m)$ to denote this mapping.

$$\zeta(m) = \bar{n} \text{ if VNF } m \text{ is attached to server } \bar{n}$$

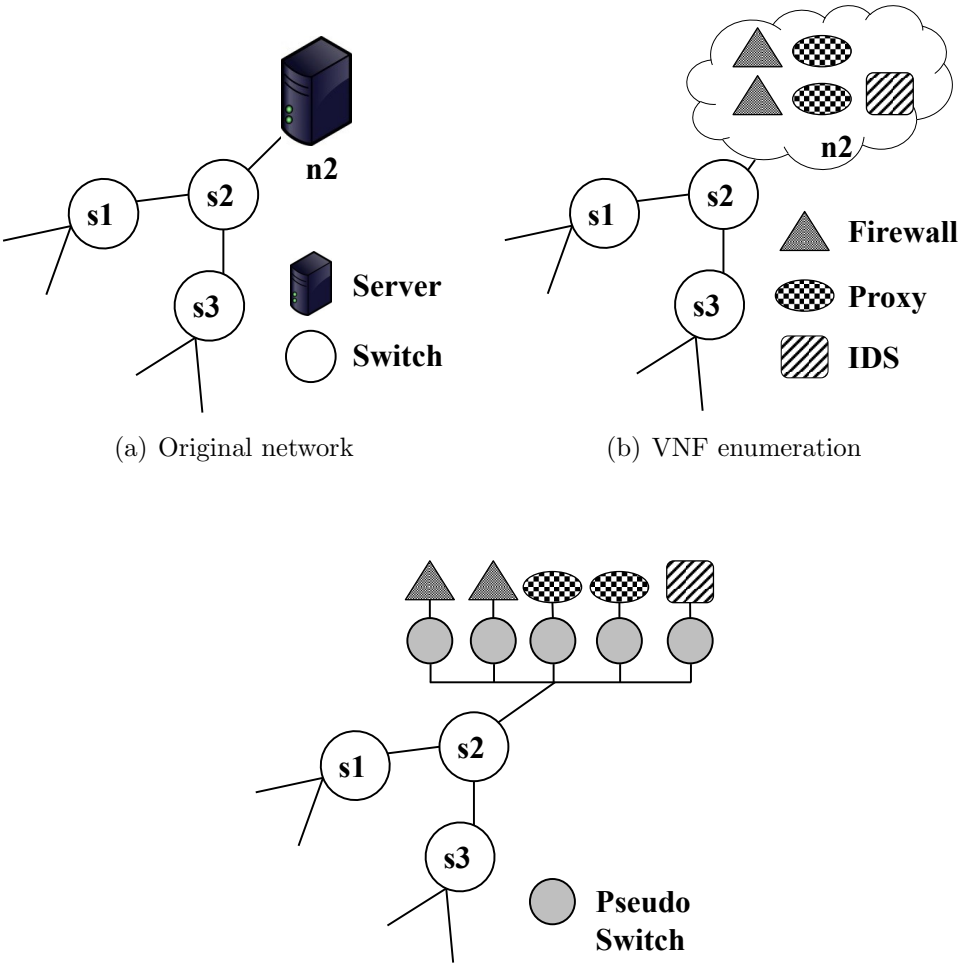
We also define a function $\Omega(\bar{n})$ to represent this mapping in the opposite direction:

$$\Omega(\bar{n}) = \{m \mid \zeta(m) = \bar{n}\},\ m \in \mathcal{M}, \bar{n} \in \bar{N}$$

Next, we define $q_{mp} \in \{0, 1\}$ to indicate the type of a VNF:

$$q_{mp} = \begin{cases} 1 & \text{if VNF } m \text{ is of type } p \in P, \\ 0 & \text{otherwise.} \end{cases}$$

We also define the function $\tau(m)$ that returns the type of pseudo-VNF m:



(a) Original network

(b) VNF enumeration

(c) Adding pseudo-switches

Figure 3.4: Network transformation

$$\tau(m) = \{p \mid q_{mp} = 1\},\ m \in \mathcal{M}, p \in P$$

As discussed earlier, a given type of VNF can be deployed on a specific set of servers. To ensure this we must have:

$$q_{mp} = d_{\zeta(m)p} \tag{3.1}$$

We should note that pseudo-VNFs simply represent where a particular type of VNF can be provisioned. $y_m \in \{0, 1\}$ indicates whether a pseudo-VNF is active or not.

$$y_m = \begin{cases} 1 & \text{if pseudo-VNF } m \in \mathcal{M} \text{ is active,} \\ 0 & \text{otherwise.} \end{cases}$$

Adding Pseudo-Switches

Next, we augment the physical topology again by adding a pseudo-switch between each pseudo-VNF and the original switch to which it was connected. This process is shown in Figure 3.4(c). We perform this step to simplify the expressions of the network flow conservation constraint in the ILP formulation. This process does not increase the size of the solution space as we consider them only for the flow conservation constraint.

3.4.2 ILP Formulation

We define the decision variable x^t_{nm} to represent the mapping of a traffic node to a pseudo-VNF:

$$x^t_{nm} = \begin{cases} 1 & \text{if node } n \in N^t \text{ is provisioned on } m \in \mathcal{M}, \\ 0 & \text{otherwise.} \end{cases}$$

Next, we define another variable to represent the mapping between a traffic node and a switch in the physical network.

Table 3.1: Glossary of symbols

Physical Network	
$\bar{G}(\bar{S}, \bar{L})$	Physical network $\bar{G}$ with switches $\bar{S}$ and links $\bar{L}$
$\bar{N}$	Set of servers
$\bar{h}_{\bar{n}\bar{s}} \in \{0,1\}$	If server $\bar{n} \in \bar{N}$ is attached to switch $\bar{s} \in \bar{S}$
R	Set of resources offered by servers
$c^r_{\bar{n}} \in \mathbb{R}^+$	Resource capacity of server $\bar{n},\ \forall r \in R$
$\beta_{\bar{u}\bar{v}}, \delta_{\bar{u}\bar{v}} \in \mathbb{R}^+$	Bandwidth, propagation delay of link $(\bar{u}, \bar{v}) \in \bar{L}$
$\eta(\bar{u})$	Neighbors of switch $\bar{u}$
$a_{\bar{n}} \in \{0,1\}$	$a_{\bar{n}} = 1$ if Server $\bar{n}$ is active
$f_{\bar{u}\bar{v}} \in \{0,1\}$	$f_{\bar{u}\bar{v}} = 1$ if physical link $(\bar{u}, \bar{v})$ is active
Virtualized Network Functions (VNFs)	
P	Set of possible VNF types
$\mathcal{D}^+_p, \kappa^r_p, c_p, \delta_p$	Deployment cost, resource requirement, processing capacity and processing delay of VNF type $p \in P$
$d_{\bar{n}p} \in \{0,1\}$	$d_{\bar{n}p} = 1$ if VNF type p can be provisioned on server $\bar{n}$
Traffic	
$\bar{u}^t, \bar{v}^t, \Psi^t$	Ingress, egress and VNF sequence for traffic t
$\beta^t, \delta^t, \omega^t$	Bandwidth, expected delay, SLA penalty for t
N^t	$\{\bar{u}^t, \bar{v}^t, \Psi^t\}$
L^t	$\{(\bar{u}^t, \Psi^t_1), \ldots, (\Psi^t_{\lvert\Psi^t\rvert-1}, \Psi^t_{\lvert\Psi^t\rvert}), (\Psi^t_{\lvert\Psi^t\rvert}, \bar{v}^t)\}$
$\eta^t(n)$	Neighbors of $n \in N^t$
$g^t_{np} \in \{0,1\}$	$g^t_{np} = 1$ if node $n \in N^t$ is of type $p \in P$
$\mathcal{M}$	Set of pseudo-VNFs
$\zeta(m)$	$\zeta(m) = \bar{n}$ if VNF $m \in \mathcal{M}$ is attached to server $\bar{n}$
$\Omega(\bar{n})$	$\{m \mid \zeta(m) = \bar{n}\},\ m \in \mathcal{M}, \bar{n} \in \bar{N}$
$q_{mp} \in \{0,1\}$	$q_{mp} = 1$ if VNF $m \in \mathcal{M}$ is of type $p \in P$
Decision Variables	
$^*x^t_{nm} \in \{0,1\}$	$x^t_{nm} = 1$ if node $n \in N^t$ is provisioned on $m \in \mathcal{M}$
$^*w^{tn_1n_2}_{\bar{u}\bar{v}} \in \{0,1\}$	$w^{tn_1n_2}_{\bar{u}\bar{v}} = 1$ if $(n1, n2) \in L^t$ uses physical link $(\bar{u}, \bar{v}) \in \bar{L}$
Derived Variables	
$^*y_m \in \{0,1\}$	$y_m = 1$ if VNF $m \in \mathcal{M}$ is active
$z^t_{n\bar{s}} \in \{0,1\}$	$z^t_{n\bar{s}} = 1$ if node $n \in N^t$ is attached to switch $\bar{s}$
$^*\hat{x}^t_{nm}, \hat{w}^{tn_1n_2}_{\bar{u}\bar{v}}, \hat{y}_m$ denote value from the previous iteration	

$$z^t_{n\bar{s}} = \begin{cases} 1 & \text{if node } n \in N^t \text{ is attached to switch } \bar{s}, \\ 0 & \text{otherwise.} \end{cases}$$

$z^t_{n\bar{s}}$ is not a decision variable as it can be derived from x^t_{nm}:

$$z^t_{n\bar{s}} = 1 \text{ if } x^t_{nm} = 1 \text{ and } \bar{h}_{\zeta(m)\bar{s}} = 1$$

We can also derive the variable y_m from x^t_{nm} as follows:

$$y_m = 1 \text{ iff } \sum_{t \in T} \sum_{n \in N^t} x^t_{nm} > 0$$

We assume that $\hat{x}^t_{nm}$ represents the value of x^t_{nm} at the last traffic provisioning event. To ensure that resources for previously provisioned traffic are not deallocated we must have $x^t_{nm} \geq \hat{x}^t_{nm},\ \forall\ t \in \hat{T}, n \in N^t, m \in \mathcal{M}$. Now, we define $\hat{y}_m \in \{0, 1\}$ that represents the value of y_m at the last traffic provisioning event as follows:

$$\hat{y}_m = 1 \text{ iff } \sum_{t \in T} \sum_{n \in N^t} \hat{x}^t_{nm} > 0$$

Again, to ensure that resources for previously provisioned traffic are not deallocated we must have $y_m \geq \hat{y}_m,\ \forall\ m \in \mathcal{M}$. Next, we need to ensure that VNF capacities are not over-committed. The processing capacity of an active VNF must be greater than or equal to the total amount of traffic passing through it. We express this constraint as follows:

$$\sum_{t \in T} \sum_{n \in N^t} x^t_{nm} \times \beta^t \leq c_{\tau(m)}, \forall\ m \in \{a | a \in \mathcal{M}, y_a = 1\} \tag{3.2}$$

We also need to make sure that physical server capacity constraints are not violated by the deployed VNFs. We represent this constraint as follows:

$$\sum_{m \in \Omega(\bar{n})} y_m \times \kappa_m^r \leq c_{\bar{n}}^r, \ \forall \ \bar{n} \in \bar{N}, r \in R \tag{3.3}$$

Each node of a traffic must be mapped to a proper VNF type. This constraint is represented as follows:

$$x_{nm}^t \times g_{np}^t = q_{mp}, \ \forall \ t \in T, n \in N^t, m \in \mathcal{M}, p \in P \tag{3.4}$$

Next, we need to ensure that every traffic node is provisioned and to exactly one VNF.

$$\sum_{t \in T} \sum_{n \in N^t} x_{nm}^t = 1, \ \forall \ m \in \mathcal{M} \tag{3.5}$$

Now, we define our second decision variable to represent the mapping between links in the VNF chain (Figure 3.3) to the links in the physical network.

$$w_{\bar{u}\bar{v}}^{tn_1n_2} = \begin{cases} 1 & \text{if } (n1, n2) \in L^t \text{ uses physical link } (\bar{u}, \bar{v}), \\ 0 & \text{otherwise.} \end{cases}$$

We also assume that $\hat{w}_{\bar{u}\bar{v}}^{tn_1n_2}$ represents the value of $w_{\bar{u}\bar{v}}^{tn_1n_2}$ at the last traffic provisioning event. To ensure that resources for previously provisioned traffic are not deallocated in the current iteration we must have

$$\begin{aligned} w_{\bar{u}\bar{v}}^{tn_1n_2} \geq \hat{w}_{\bar{u}\bar{v}}^{tn_1n_2}, \forall \ t \in \hat{T}, \forall (n_1, n_2) \in \{(a, b) | \\ a \in N^t, b \in \eta^t(a), b > a\}, \forall \bar{u}, \bar{v} \in \bar{S} \end{aligned} \tag{3.6}$$

To ensure that each directed link in a traffic request is not mapped to both directions of a physical link, we must have:

$$\begin{aligned} w_{\bar{u}\bar{v}}^{tn_1n_2} + w_{\bar{v}\bar{u}}^{tn_1n_2} \leq 1, \forall \ t \in T, \forall (n_1, n_2) \in \{(a, b) | \\ a \in N^t, b \in \eta^t(a), b > a\}, \forall \bar{u}, \bar{v} \in \bar{S} \end{aligned} \tag{3.7}$$

Now, we present the capacity constraint for physical links:

$$\sum_{\bar{u}\in\bar{S}}\sum_{\bar{v}\in\bar{S}}(w_{\bar{u}\bar{v}}^{tn_1n_2}+w_{\bar{v}\bar{u}}^{tn_1n_2})\times\beta^t\leq\beta_{\bar{u}\bar{v}},\forall\ t\in T,$$
$$\forall(n_1,n_2)\in\{(a,b)|a\in N^t,b\in\eta^t(a),b>a\} \tag{3.8}$$

Next, we present the flow constraint that makes sure that the in-flow and out-flow of each switch in the physical network is equal except at the ingress and egress switches:

$$\sum_{\bar{v}\in\eta(\bar{u})}\left(w_{\bar{u}\bar{v}}^{tn_1n_2}-w_{\bar{v}\bar{u}}^{tn_1n_2}\right)=z_{n_1\bar{u}}^t-z_{n_2\bar{u}}^t,\forall\ t\in T,$$
$$\forall(n_1,n_2)\in\{(a,b)|a\in N^t,b\in\eta^t(a),b>a\},\forall\bar{u}\in\bar{S} \tag{3.9}$$

Finally, we need to ensure that every link in a traffic request is provisioned on a path in the physical network:

$$\sum_{\bar{u}\in\bar{S}}\sum_{\bar{v}\in\bar{S}}(w_{\bar{u}\bar{v}}^{tn_1n_2}+w_{\bar{v}\bar{u}}^{tn_1n_2})\geq 0,\forall\ t\in T,$$
$$\forall(n_1,n_2)\in\{(a,b)|a\in N^t,b\in\eta^t(a),b>a\} \tag{3.10}$$

Our objective is to find the optimal number and placement of VNFs that minimizes OPEX and physical resource fragmentation in the network. We formulate them as follows:

OPEX: We consider the following four cost components contributing to OPEX:

1. VNF Deployment Cost: the VNF deployment cost can be calculated based on three different scenarios. Let us assume that we need to deploy a VNF of type p on server $\bar{n}$. Now one of the following three cases may occur:

- **Case 1:** There is no VNF of type p on $\bar{n}$, so, we need to deploy a new one.
- **Case 2:** There is a VNF of type p on $\bar{n}$, but it does not have enough residual capacity to support the new traffic. So, we need to deploy a new VNF.

- **Case 3:** There is a VNF of type p on $\bar{n}$ and it has enough residual capacity to support the new traffic. So, there is no need to deploy a new VNF.

Considering all three scenarios, the total VNF deployment cost can be expressed as follows:

$$\mathbb{D} = \sum_{m \in \mathcal{M} | y_m = 1} \mathcal{D}_p^+ \times q_{mp} \times (y_m - \hat{y}_m) \tag{3.11}$$

2. Energy Cost: Without loss of generality we assume that the energy consumption of a server is proportional to the amount of resources being used. However, a server usually consumes power even in the idle state. So, we compute the power consumption of a server as follows:

$$\mathbb{E}_{\bar{n}} = \sum_{m \in \Omega \bar{n}} y_m \times q_{mp} \times e^r(c^r_{\bar{n}}, \kappa^r_p)$$

where

$$e^r(r_t, r_c) = (e^r_{max} - e^r_{idle}) \times \frac{r_c}{r_t} + e^r_{idle}$$

Here, r_t and r_c denote the total and consumed resource, respectively. e^r_{idle} and e^r_{max} denote the energy cost in the idle and peak consumption states for resource r, respectively.

Hence, the total energy cost is

$$\mathbb{E} = \sum_{\bar{n} \in \bar{N}} \sum_{m \in \Omega \bar{n}} y_m \times q_{mp} \times e^r(c^r_{\bar{n}}, \kappa^r_p) \tag{3.12}$$

3. Cost of Forwarding Traffic: Let us assume that the cost of forwarding 1 Mbit data through one link in the network is σ (in dollars). We can compute the total cost of traffic forwarding as follows:

$$\mathbb{F} = \sum_{t \in T} \sum_{n_1 \in N^t} \sum_{\substack{n_2 \in \eta^t(n_1) \\ \text{and } n_2 > n_1}} \sum_{\bar{u} \in \bar{S}} \sum_{\bar{v} \in \eta(\bar{u})} \Big((w_{\bar{u}\bar{v}}^{tn_1n_2} - \hat{w}_{\bar{u}\bar{v}}^{tn_1n_2}) \times \beta^t \times \sigma \Big) \tag{3.13}$$

4. Penalty for SLA violation: We can compute the actual propagation delay experienced by a traffic as follows:

$$\delta_t^a = \sum_{n_1 \in N^t} \sum_{\substack{n_2 \in \eta^t(n_1) \\ \text{and } n_2 > n_1}} \sum_{\bar{u} \in \bar{S}} \sum_{\bar{v} \in \eta(\bar{u})} w_{\bar{u}\bar{v}}^{tn_1n_2} \delta_{\bar{u}\bar{v}}$$

Let $\rho^t(\omega^t, \delta^t, \delta_a^t)$ be a function that computes the penalty for SLO violation given the policy for determining penalty (ω^t), expected propagation delay (δ^t) and actual propagation delay (δ_a^t) for traffic t. So, the total cost for SLA violations can be expressed as follows:

$$\mathbb{P} = \sum_{t \in T} \rho^t(\omega^t, \delta_t, \delta_a^t) \tag{3.14}$$

– **Resource Fragmentation:** Our second objective is to minimize resource (*e.g.*, server and link) fragmentation of active servers and links. We express it using the same unit as the above mentioned costs. For this purpose, we assume that p^r denotes the unit price resource of type $r \in R$. We also denote ρ^β as the unit price bandwidth.

A physical server $\bar{n}$ is considered active if it hosts at least one active pseudo-VNF. The binary variable $a_{\bar{n}}$ captures this property:

$$a_{\bar{n}} = \begin{cases} 1 & \text{if } \sum_{m \in \Omega(\bar{n})} y_m > 0, \\ 0 & \text{otherwise.} \end{cases}$$

Similarly, a physical link $(\bar{u}, \bar{v})$ is considered active if it is hosting at least one traffic flow. We use the binary variable $f_{\bar{u}\bar{v}}$ to represent this:

$$f_{\bar{u}\bar{v}} = \begin{cases} 1 & \text{if} \quad \sum\limits_{\substack{t \in T, \\ (n_1,n_2) \in L^t}} w_{\bar{u}\bar{v}}^{tn_1n_2} > 0, \\ 0 & \text{otherwise.} \end{cases}$$

Now, we can compute the total cost for resource fragmentation as follows:

$$\begin{aligned} \mathbb{C} = \sum_{\bar{n} \in \bar{N}} a_{\bar{n}} \sum_{r \in R} \Big(c_{\bar{n}}^r - \sum_{m \in \Omega(\bar{n})} (\kappa_p^r \times q_{mp} y_m) \Big) p^r + \\ \sum_{\bar{u} \in \bar{S}} \sum_{\bar{v} \in \eta(\bar{u})} f_{\bar{u}\bar{v}} \Big(\beta_{\bar{u}\bar{v}} - \sum_{t \in T} \sum_{n_1 \in N^t} \sum_{\substack{n_2 \in \eta^t(n_1) \\ \text{and } n_2 > n_1}} \\ (w_{\bar{u}\bar{v}}^{tn_1n_2} \times \beta^t) \Big) \rho^\beta \end{aligned} \tag{3.15}$$

Here, the first term represents the cost of server resource fragmentation (*e.g.*, CPU, memory, and disk) and the second term represents the cost of link bandwidth fragmentation.

Our objective is to minimize the total network operational cost and resource fragmentation that can be expressed as a weighted sum of the aforementioned costs.

$$\text{minimize } \Big(\alpha \mathbb{D} + \beta \mathbb{E} + \gamma \mathbb{F} + \lambda \mathbb{P} + \mu \mathbb{C} \Big) \tag{3.16}$$

Where α, β, γ, λ and μ are weighting factors used to adjust the relative importance of the cost components.

VNF-OP is NP-Hard. We reduce the NP-Hard Capacitated Plant Location Problem with Single Source constraints (CPLPSS) [106] to the VNF-OP. In CPLPSS, we are given a set of potential locations for production plants with fixed costs and capacities. A commodity produced by these plants is to be supplied to a set of customers with fixed demands and associated transportation costs. Moreover, each customer must be served by a single plant. The objective is to find a subset of the plants that should be operated to minimize cost without violating capacity and demand constraints.

Given an instance of the CPLPSS we can transform it to an instance of VNF-OP in the following manner: (i) for each customer we create the chain `DS` $\rightarrow$ `plant` $\rightarrow$ `customer`,

where DS is a dummy ingress switch, customer is the egress switch, and plant is a VNF, (ii) set the bandwidth of the chain to be equal to the customer demand, (iii) use the transportation cost as the traffic forwarding cost, (iv) configure each physical machine to deploy a single VNF of type plant, and (V) set the processing capacity of each plant to be equal to its production capacity. These operations can be performed in polynomial time of the problem size. Now, if we can solve this instance of VNF-OP, we will also obtain a solution for the CPLPSS. However, CPLPSS is NP-hard, so the VNF-OP is NP-hard as well.

3.5 Heuristic Solution

In this section, we present a heuristic to solve the VNF-OP. Given a network topology, a set of middlebox specifications and a batch of traffic requests, the heuristic finds the number and locations of different types of VNFs required to operate the network with minimal OPEX. We did not explicitly consider resource fragmentation to keep the heuristic simple and fast. However, our experimental results show that even with this simplification, the heuristic produces solutions that are very close to the optimal. The heuristic runs in two steps. First, we model the VNF-OP as a multi-stage directed graph with associated costs. Then we find a near-optimal VNF placement from the multi-stage graph by running the Viterbi algorithm [107]. In the following, we first describe the modeling of VNF-OP using multi-stage graph (Section 3.5.1), followed by the solution using Viterbi algorithm (Section 3.5.3).

3.5.1 Modeling with Multi-Stage Graph

For a given traffic request, $t = \langle \bar{u}^t, \bar{v}^t, \Psi^t, \beta^t, \delta^t, \omega^t \rangle$, we represent t as a multi-stage graph with $l_{\Psi^t} + 2$ stages. The first and the last (*i.e.*, $l_{\Psi^t} + 2$) stages represent the ingress and egresses switches, respectively. These two stages contain only one node representing $\bar{u}^t$ and $\bar{v}^t$, respectively. Stage i ($\forall i \in \{2, \dots (l_{\Psi^t} + 1)\}$), represents the $(i-1)$-th VNF in the traffic request and the node(s) within this stage represent the possible server locations where that type of VNFs can be placed. Each node is associated with a VNF deployment cost (Eq. 3.11) and an energy cost (Eq. 3.12) as described in Section 3.4.2.

An edge $(\bar{v}_i, \bar{v}_j)$ in this multi-stage graph represents the placement of a VNF at a server attached to switch $\bar{v}_j$, given that the previous VNF in the sequence is deployed on a server attached to switch $\bar{v}_i$. We add a directed edge between all pairs of nodes in stage i and $i+1$

$(\forall i \in \{1, 2, \ldots (l_{\Psi^t} + 1)\})$. We associate two costs with each edge: the cost of forwarding traffic (Eq. 3.13) and the penalty for SLA violations (Eq. 3.14). The traffic forwarding cost is proportional to the weighted shortest path (in terms of latency) between the switches. The penalty for SLA violations is obtained by the following process: (i) we equally divide the maximum allowed delay between the stages, (ii) we assign a SLA violation cost for a transition between two successive stages in the multi-stage graph whenever we incur more than the allocated delay due to traffic transport and processing at the nodes. The total cost of a transition between two successive stages is calculated by summing the node and edge costs following Eq. 3.16. Finally, a path from the node in the first stage to the node in the last stage represents a placement of the VNFs. Our goal is to find a path in the multi-stage graph that yields minimal OPEX.

3.5.2 Heuristic Algorithm

Algorithm 3 gives the pseudcode of the heuristic solution. The procedure $ProvisionTraffic$ takes as input a traffic request t and the network topology graph $\bar{G}$ annotated with the resource capacities at each switch. We keep two tables, *cost* and π, to keep track of the cost and the sequence of middlebox placements, respectively. $cost_{i,j}$ represents the cost of deploying the j-th middlebox in the middlebox sequence Ψ^t to a server attached with switch i. The cost computation procedure is the same as described in Section 3.5.3. We use a number of helper procedures for the ease of implementation. The first helper procedure, $IsResourceAvailable$ checks if a middlebox $mbox$ for a traffic request t can be placed at switch i, satisfying the minimum bandwidth and resource requirements. The second helper, $GetCost$, computes the cost of placing middlebox $mbox$ for a traffic request t at a server attached to switch j. The previous node k that yields the minimum cost for the current node in consideration j, is tracked by the entry $\pi_{k,j}$. Finally, we backtrace using entries in π to obtain the desired middlebox sequence.

Running Time: Let the number of switches and the maximum length of a middlebox sequence be n and m, respectively. Algorithm 3 performs $\Theta(nm)$ computations at the beginning to initialize the cost matrix. Then for each element in the traffic sequence, the algorithm takes all possible pairs of nodes u, v and computes the cost of deploying a middlebox at the server attached to switch v given that the previous middlebox in the sequence was deployed at a server connected to switch u. Therefore, there is a total of $\Theta(n^2m)$ operations involved. With some pre-computation steps the costs can be calculated and resource availability can be queried in $O(1)$ time. Therefore, Algorithm 3 runs in $\Theta(n^2m)$.

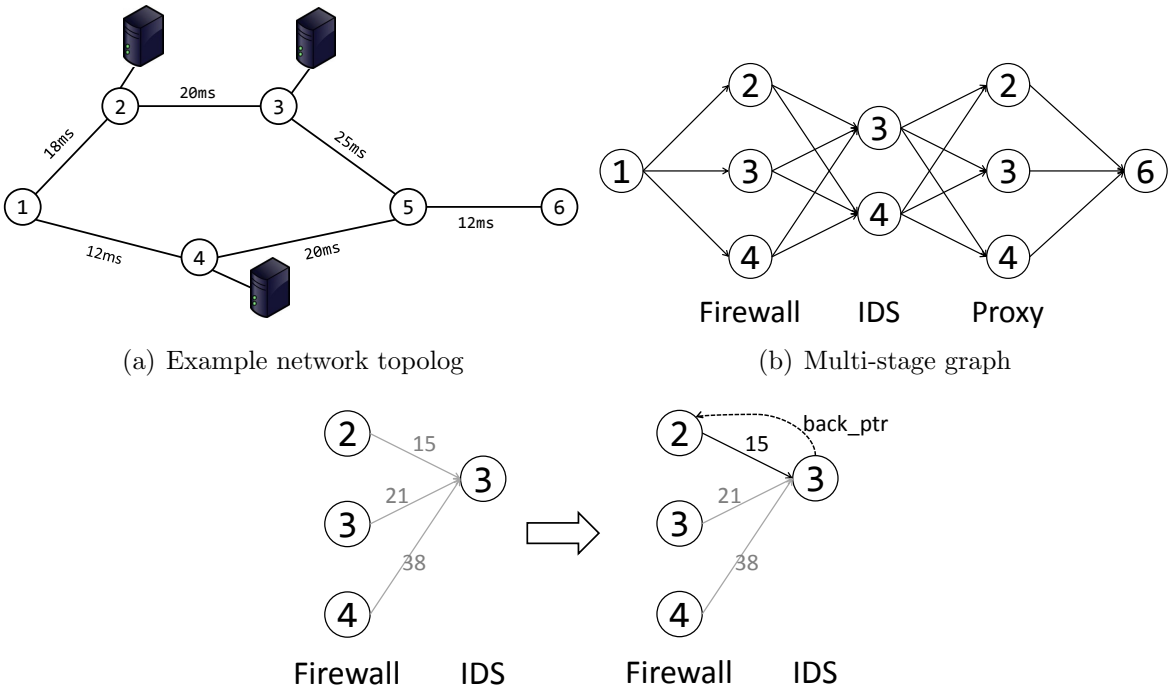
Algorithm 3 ProvisionTraffic(t, $\bar{G}$)

$\forall(i,j) \in \{1 \dots |\Psi^t|\} \times \{1 \dots |\bar{S}|\} : cost_{i,j} \leftarrow \infty$, $\pi_{i,j} \leftarrow NIL$
$\forall i \in |\bar{S}| :$
if $IsResourceAvailable(u^t, i, \Psi_1^t, t)$ **then**
  $cost_{1,n} \leftarrow GetCost(u^t, i, \Psi_1^t, t)$, $\pi_{1,n} \leftarrow n$
end if
$\forall(i,j,k) \in \{2 \dots |\Psi^t|\} \times \{1 \dots |\bar{S}|\} \times \{1 \dots |\bar{S}|\} :$
if $IsResourceAvailable(k, j, \Psi_i^t, t)$ **then**
  $cost_{i,j} \leftarrow \min\{cost_{i,j}, cost_{i-1,k} + GetCost(k, j, \Psi_i^t, t)\}$
  $\pi_{i,j} \leftarrow i$ yielding minimum $cost_{i,j}$
end if
$\Pi \leftarrow NIL$, $C \leftarrow \infty$, $\psi \leftarrow <>$
$\forall i \in |\bar{S}| :$
  $C \leftarrow \min\{C, cost_{|\Psi^t|,i} + ForwardingCost(i, v^t) + SLAViolationCost(i, v^t, t)\}$
  $\Pi \leftarrow i$ yielding minimum $cost_{|\Psi^t|,i}$
$\forall i \in < |\Psi^t|, |\Psi^t| - 1 \dots 1 > :$ Append Π to ψ, $\Pi \leftarrow \pi_{i,\Pi}$
return $Reverse(\psi)$

3.5.3 Finding a Near-Optimal Solution

Viterbi algorithm is a widely used method for finding the most likely sequence of states from a set of observed states. To find such a sequence, Viterbi algorithm first models the states and their relationships as a multi-stage graph. Each stage consists of the possible states and a transition cost is assigned between all pairs of states in successive stages. Once the multi-stage graph is constructed, Viterbi algorithm proceeds by computing a per node cumulative cost, $cost_u$. This cost is computed recursively as the minimum of $cost_v + transition\_cost(v, u)$, for all v in the previous stage as of u's stage. $cost_u$ represents the cost of including node u in the final solution. This computation proceeds in the increasing order of stages. After finishing the computation at the final stage, the most likely sequence of states is constructed by tracing back a path from the final stage back to the first that yields the minimum cost.

We borrow the idea of how costs are computed from Viterbi Algorithm and propose a traffic provisioning algorithm, $ProvisionTraffic$ (Algorithm 3). It takes a traffic request t and a network topology $\bar{G}$ as input and returns a placement of Ψ^t in $\bar{G}$. For each node u in each stage i, we find a node v in stage $i-1$ that yields the minimum total cost



(a) Example network topolog

(b) Multi-stage graph

(c) A single stage

Figure 3.5: Modeling with multi-stage graph

$cost_{v,u}$ (costs are defined according to the discussion in Section 3.5.1). We keep track of the minimum cost path using the table π. After finishing computation for the final stage, we construct the desired VNF placement by back tracing from the final stage to the first stage, using the entries in π. During this process we update residual resource capacities of the servers and the residual bandwidth of the links after each path is allocated. For each traffic request, the heuristic solution runs in $\Theta(n^2m)$ time, where n is the number of switches in the network and m is the VNF sequence length.

3.5.4 Heuristic in Action

Figure 3.5(a) shows an example network topology with six switches, where the servers are connected to switch 2, 3 and 4. We need to find the path for a traffic which is going from switch 1 to 6 and must pass through a firewall, then an IDS and finally through a proxy.

First, we generate a multi-stage graph as shown in Figure 3.5(b). Here, we are assuming

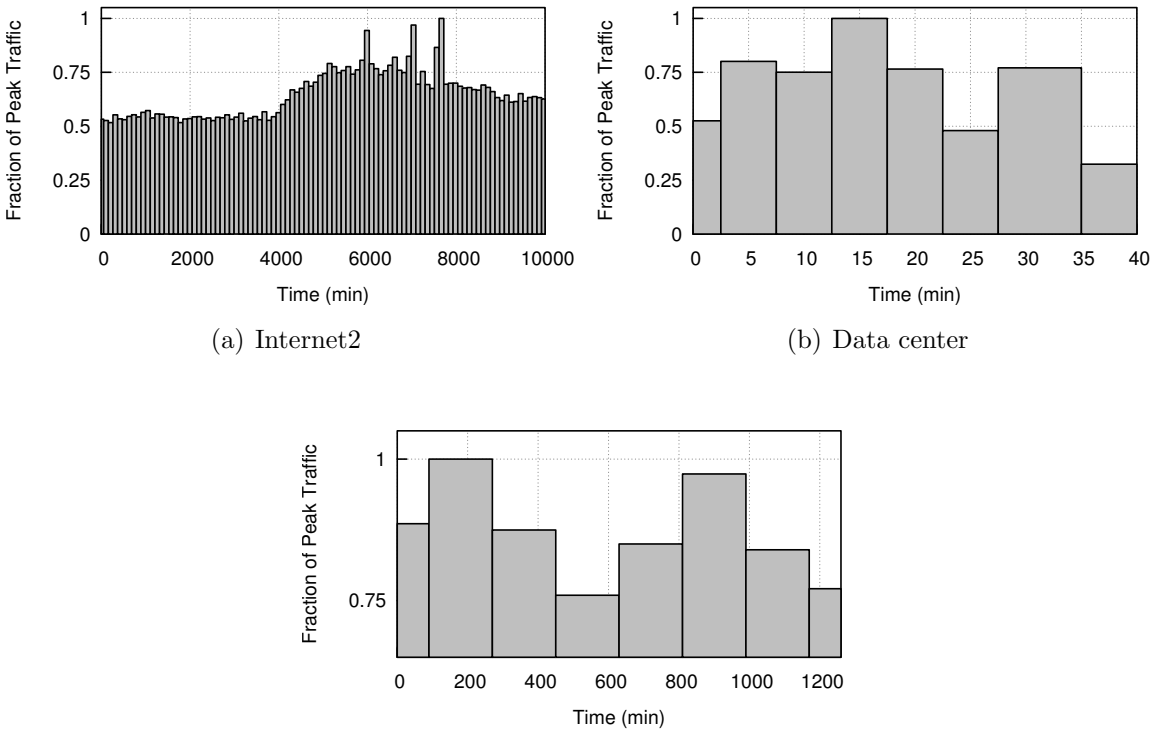
that the firewall and proxy can be deployed on any server, but the IDS can only be deployed on servers connected to switches 3 and 4. Each node in the multi-stage graph represents a decision about where to place a VNF. For example, if we select node 4 in the stage labeled "IDS", it means that a VNF corresponding to an IDS will be deployed on the server connected to switch 4. As explained earlier, there is a cost associated with each node selection.

Now, we traverse this graph starting at node 1. The first stage is trivial, we just compute the cost of deploying and running (energy cost) a firewall at node 2, 3 and 4 and add the cost of routing traffic from node 1 to each node. There is no additional computation as there is just one incoming link for each node. However, the operations for the subsequent stages involve comparing the cost of reaching a particular node from different nodes. For example, node 3 in stage "IDS" can be reached from three different nodes. The operation performed in this is stage is explained in Figure 3.5(c).

We need to compute the transition cost from nodes 2, 3 and 4 to node 3. These costs are shown on the left side of Figure 3.5(c). Now, if we select the link between node 4 and node 3 then the Firewall will be deployed on node 4 and the IDS will be deployed on node 3 and the cost of deploying the IDS will be 38. However, we have links with lower costs than this one and at each stage we select the incoming link with the minimal cost. So, here we will select the link between node 2 and 3 as it has the lowest cost of 15. We will also save a pointer (`back_ptr`) to mark the node that was selected. We continue in this manner until we reach the destination node (node 6 in this example), then we follow the `back_ptr`s to re-construct the VNF-to-server mappings.

3.6 Performance Evaluation

We perform trace driven simulations on real-world network topologies to gain a deeper insight, and to evaluate the effectiveness of the proposed solution. Our simulation is focused on the following aspects: (i) demonstrating the benefits of dynamic VNF orchestration over hardware middleboxes (Section 3.6.3), (ii) comparing the performance of the heuristic solution with that of the CPLEX based optimal solution (Section 3.6.4), (iii) comparing the performance of our heuristic with state-of-the art (Section 3.6.5), (iv) demonstrating the scalability of our heuristic (Section 3.6.6), and (v) analyzing the behavior of the proposed solution for different traffic volumes (Section 3.6.7). Before presenting the results, we briefly describe the simulation setup (Section 3.6.1) and the performance metrics (Section 3.6.2). Implementations of both CPLEX and heuristic are available at `http://goo.gl/Da7EZu`.



(a) Internet2

(b) Data center

(c) Rocketfuel

Figure 3.6: Traffic distribution over time for different scenarios

3.6.1 Simulation Setup

Topology Dataset

We have used a wide range of network topologies: (i) Internet2 research network (12 nodes, 15 links) [66], (ii) A university data center network (23 nodes, 42 links) [108] and (iii) Autonomous System 3967 (AS-3967) from Rocketfuel topology dataset (79 nodes, 147 links) [109].

Traffic Dataset

We use both real traces and synthetically generated traffic for the evaluation. We use traffic matrix traces from [66] to generate time varying traffic for the Internet2 topology. This

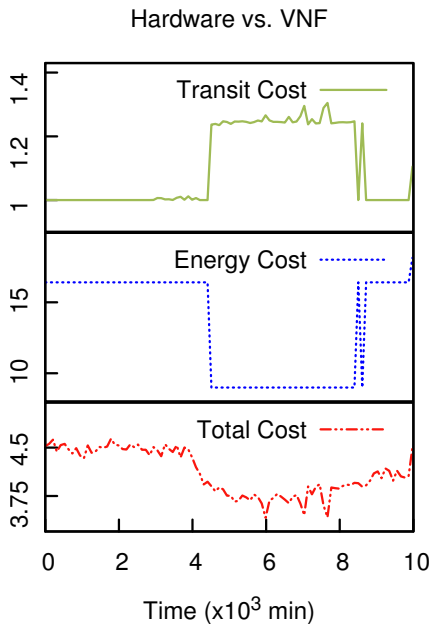
Table 3.2: Server and middlebox data used in evaluation

Server Data [102]		
Physical CPU Cores	Idle Energy	Peak Energy
16	80.5W	2735W
Hardware Middlebox Data		
Idle Energy	Peak Energy	Processing Capacity
1100W	1700W	40Gbps
VNF Data [113, 45]		
Network Function	CPU Required	Processing Capacity
Firewall	4	900Mbps
Proxy	4	900Mbps
Nat	2	900Mbps
IDS	8	600Mbps

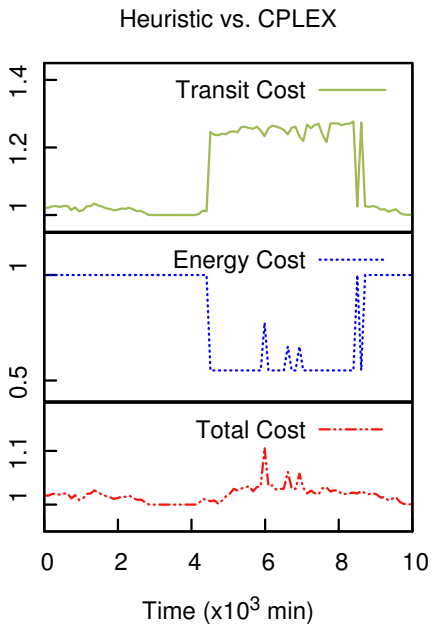
trace contains a snapshot of a 12×12 traffic matrix and demonstrates significant variation in traffic volume. For the data center network, we use the traces available from [108], and replay the traffic between random source-destination pairs. Finally, for the Rocketfuel topology, we generate a synthetic time-varying traffic matrix using the FNSS tool [110]. The traffic matrix follows the distribution from [111] and exhibits time-of-day effect.

Middlebox and Cost Data

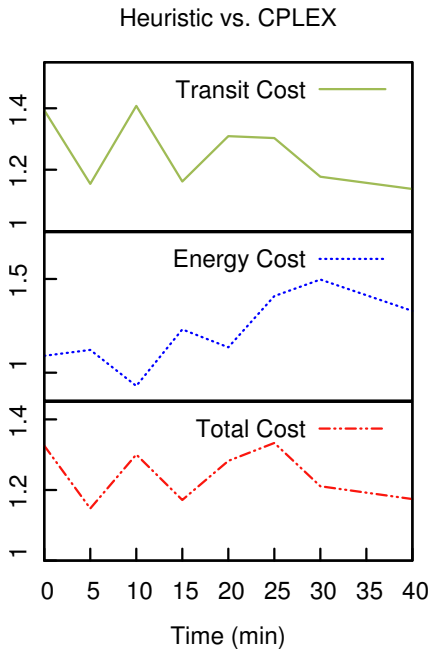
We generate a 3-length middlebox sequence for each traffic based on the data provided in [112] and [37]. We use publicly available data sheets from manufacturers and service providers to select and infer values for server energy cost, SLA violation cost (for violating maximum latency), resource requirements for software middleboxes and their processing capacities. We also obtained energy consumption data for hardware middleboxes from a popular network equipment manufacturer. Table 3.2 lists the parameters used for servers, VNFs and middleboxes. In the rest of this section we use the term "middlebox" to refer to both hardware middlebox and VNF.



(a) Hardware vs. VNF (Internet2)



(b) Internet2



(c) Data center

Figure 3.7: Time vs. cost ratio

3.6.2 Performance Metrics

Operational Expenditure (OPEX)

We measure OPEX according to Eq. 3.16, and compare CPLEX and heuristic solutions by plotting the ratio of OPEX and its components. Here, we have assigned equal weights to the cost components in Eq. 3.16.

Execution Time

It is the time required to find middlebox placement for a given traffic batch and network topology.

System Utilization

We compute it as the fraction of used CPU for a server. We also report the number of active servers.

Topological Properties of Solution

We report two topological properties of the middlebox locations: (i) percentage of middleboxes placed withing k-hops from the ingress/egress switches and (ii) *path stretch*, *i.e.*, the ratio of path length obtained by CPLEX or the heuristic to the shortest path length for the traffic. The first metric gives us an insight into the location of middleboxes with respect to the ingress/egress switches, and the second one shows how many additional links (hence more bandwidth) are required to steer traffic through middlebox sequences.

3.6.3 VNFs vs. Hardware Middleboxes

One of the driving forces behind NFV is that VNFs can significantly reduce a network's OPEX. Here, we provide quantifiable results to validate this claim. Figure 3.7(a) shows the ratio of OPEX for hardware middleboxes to VNFs for incoming traffic provisioning requests (about 132 requests per batch) over a period of 10000 minutes. We show two components of OPEX: energy and transit cost. There is no publicly available data that can be used to estimate the deployment cost of hardware middleboxes. So, for this experiment, we do not consider deployment cost as a component of OPEX to make the comparison fair. The

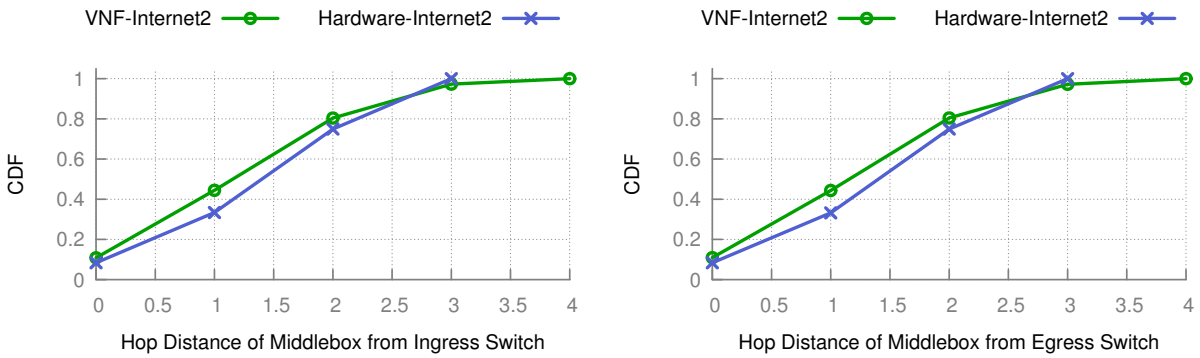
SLA violation penalty is not shown as it is zero for all time-instances. We implemented a different CPLEX program to peak provision the hardware middleboxes (peak traffic occurs at time-instance 7665). VNFs are provisioned at each time-instance by our CPLEX implementation corresponding to the formulation provided in Section 3.4.

The bottom part of Figure 3.7(a) shows that VNFs provide more than 4 times reduction in OPEX. The individual reductions in energy and transit costs are also shown in the same figure. The reduction in energy cost is much higher than that of the transit cost. This is due to the fact that hardware middleboxes consume considerably higher energy than commodity servers. From Figure 3.7(a) and Figure 3.6(a), we can also see that with the increase in traffic volume (after time-instance 4000) the total cost ratio decreases. Interestingly, the energy cost ratio decreases, but the transit cost ratio increases. Handling higher traffic volume requires higher number of VNFs to be deployed, which increases the energy consumption of commodity servers, thus decreasing the energy cost ratio. However, VNFs are provisioned at optimal locations by CPLEX, which causes the transit cost to decrease and increases the transit cost ratio. The cost ratio relationship between VNFs and hardware middleboxes depends on a number of factors like processing capacity, traffic volume, idle and peak energy consumption.

The topological properties of VNF and hardware middlebox placement locations are reported in Figure 3.8. The CDF of hop distance between the ingress switch and middlebox is shown in Figure 3.8(a). Higher percentage of VNFs are located within 2 hops of the ingress switch (mostly withing 1 hop), compared to hardware middleboxes. Some VNFs are also located at 4 hop distance. This only occurs when placing a VNF farther away reduces the OPEX by decreasing the energy cost. Similar results are obtained for the hop distance between middlebox and egress switch (Figure 3.8(b)). These two figures also demonstrate the fact that CPLEX places middleboxes in a more balanced (symmetric) way on the path between the ingress and egress switch. The path stretch for both hardware middleboxes and VNFs are shown in Figure 3.8(c). VNFs consistently achieve a lower path stretch than hardware middleboxes, as VNF locations are not static like the hardware middleboxes. They can be provisioned on any server to reduce OPEX.

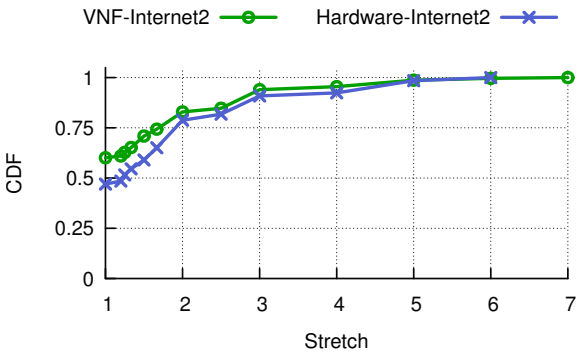
3.6.4 Performance Comparison Between CPLEX and Heuristic

Now, we compare the performance of our heuristic with that of the optimal solution. Figure 3.7(b) and Figure 3.7(c) show the cost ratios for Internet2 and data center networks, respectively. The traffic patterns for these two topologies are shown in Figure 3.6(a) and Figure 3.6(b), respectively. The deployment cost and penalty for SLO violation are



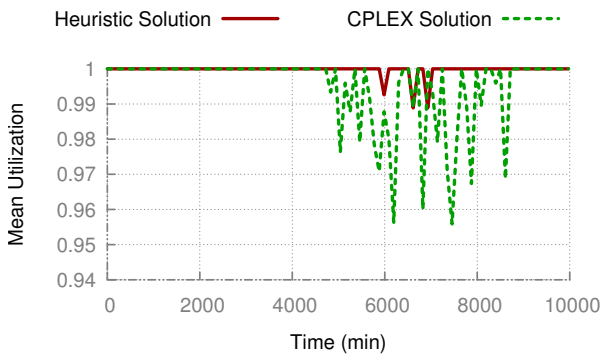
(a) Distance between middlebox and ingress

(b) Distance between middlebox and egress

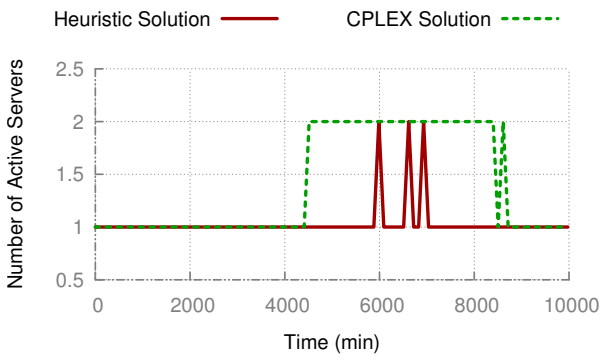


(c) Path stretch

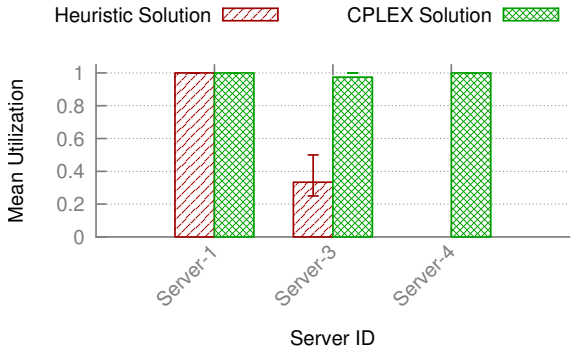
Figure 3.8: Topological property comparison between hardware middlebox and VNF deployment (Internet2)



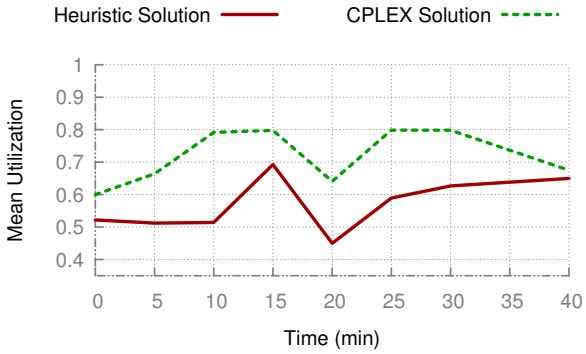
(a) Mean server utilization (Internet2)



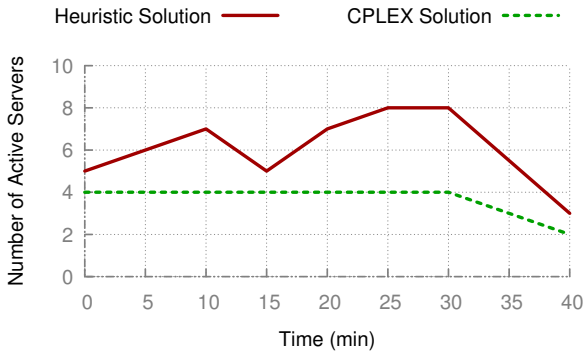
(b) Number of active servers (Internet2)



(c) Per server utilization (Internet2)



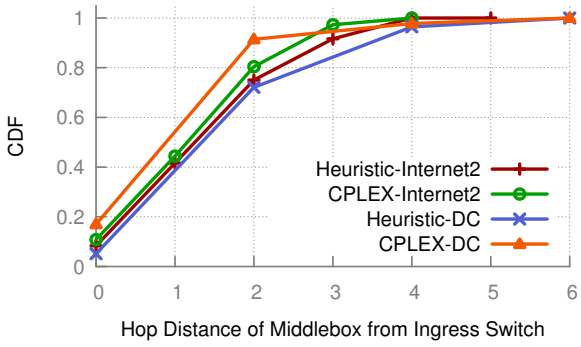
(d) Mean server utilization (data center)



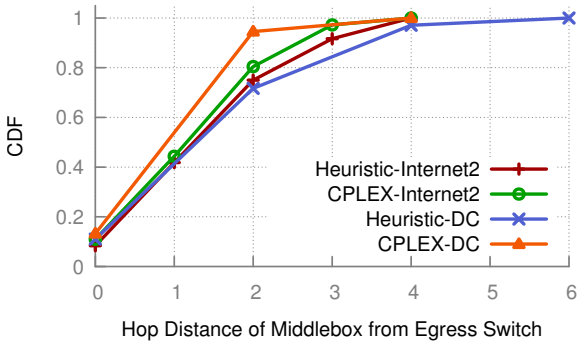
(e) Number of active servers (data center)

(f) Per server utilization (data center)

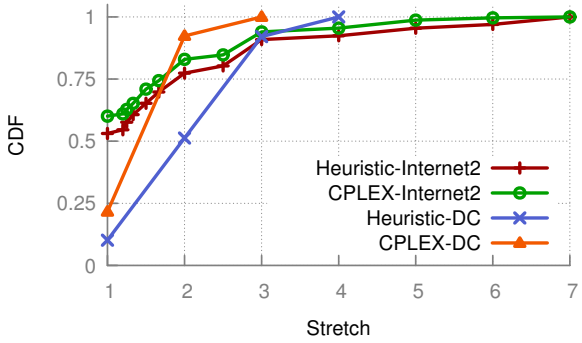
Figure 3.9: Resource utilization



(a) Distance between middlebox and ingress



(b) Distance between middlebox and egress



(c) Path stretch

Figure 3.10: Topological properties of solution

not shown, as the deployment cost is equal in both cases and the SLA violation penalty is zero for all time-instances. From Figure 3.7(b), we can see that the heuristic finds solutions that are within 1.1 times of the optimal solution. During peak traffic periods, the ratio of energy cost goes below 1, but the ratio of transit cost increases. The optimal solution adapts to high traffic volumes by deploying more VNFs (increasing energy cost) and placing them at locations that decrease the transit cost. As a result, the ratio of energy cost decreases and the ratio of transit cost increases. However, the total cost ratio stays almost the same (varying between 1 and 1.1). Similar results are obtained for the data center network (Figure 3.7(c)), where the cost ratio is also very close to 1 and varies between 1.1 and 1.3.

The average execution times of the heuristic and CPLEX are shown in Table 3.3. They were run on a machine with 10 × 16-Core 2.40GHz Intel Xeon E7-8870 CPUs and 1TB

Table 3.3: Average execution time

Topology	CPLEX	Heuristic
Internet2 (12 nodes, 15 links)	34.99s	0.535s
Data center (23 nodes, 43 links)	1595.12s	0.442s
AS-3967 (79 nodes, 147 links)	∞	2.54s

memory. As we can see, our heuristic provides solutions that are very close to the optimal one and its execution time is several orders of magnitude faster than CPLEX.

Figure 3.9 shows results related to server resource utilization for Internet2 and data center networks. Figure 3.9(a) and Figure 3.9(b) show the mean utilization and the total number of active servers, respectively, for the Internet2 topology. Figure 3.9(c) shows the average utilization per server over all time-instances. The mean utilization of the heuristic is less than that of CPLEX, as CPLEX uses more servers than the heuristic (Figure 3.9(b)). CPLEX achieves lower OPEX by deploying more VNFs during higher traffic periods to route traffic through shorter paths. However, the solutions provided by the heuristic are within 1.1 times of the optimal results (Figure 3.7(b)). In case of the data center network, CPLEX uses less servers than the heuristic (Figure 3.9(e)) and the utilization is also higher (Figure 3.9(d)). The solution provided by the heuristic has higher resource fragmentation than the CPLEX one (Figure 3.9(f)). The data center topology offers higher number of locations to deploy VNFs compared to Internet2. Hence, the heuristic falls a little short of the optimal placement as it explores a smaller solution space. CPLEX finds the optimal value, but at the cost of much higher execution time (Table 3.3).

The topological properties for middlebox deployment for Internet2 and data center networks are shown in Figure 3.10. The CDF of hop distance from the ingress switch to a VNF is shown in Figure 3.10(a). The hop distances for the heuristic is very close to that of the optimal solution. In case of the data center network, there is a relatively larger gap. This occurs due to the higher path diversity offered by a data center network. Each pair of nodes has more than one equal cost path. CPLEX finds the optimal solution by exploring all of them. However, the heuristic always picks the first shortest path. It does not explore the alternate paths to keep the execution time within practical limits (Table 3.3). Similar results are observed for the egress case (Figure 3.10(b)). From Figure 3.10(a) and Figure 3.10(b) we can also see that the CDFs are quite similar, which means that both CPLEX and heuristic place VNFs uniformly on the path between the ingress and egress switches. The path stretch is shown in Figure 3.10(c). As before, the heuristic's performance is close to that of the optimal solution. In case of the data center network, the heuristic has a larger stretch, which is a result of the path diversity inherent in data

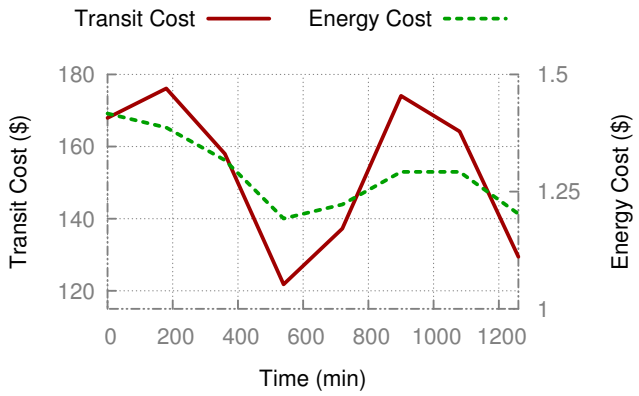


Figure 3.11: OPEX components for AS-3967

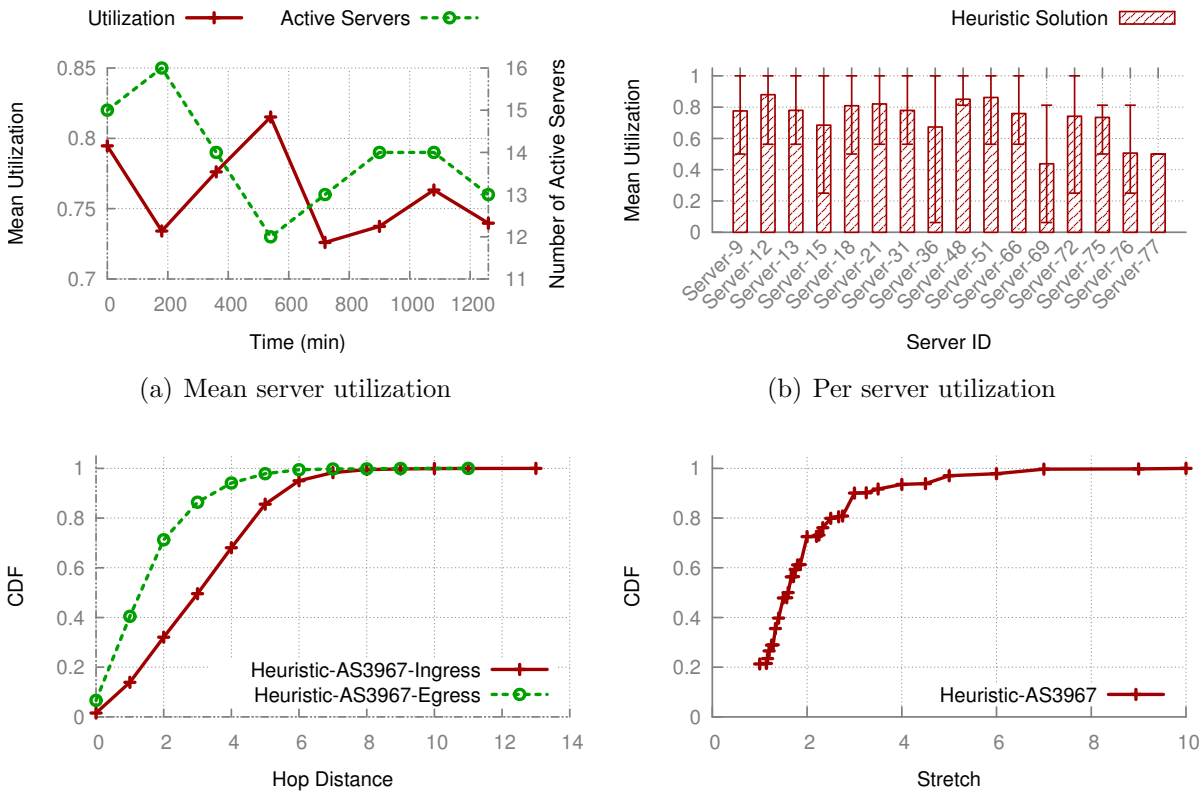
center networks.

The results for the AS-3967 topology are shown in Figure 3.11 and Figure 3.12. The traffic for this topology is shown in Figure 3.6(c). As mentioned earlier, this traffic was generated using the FNSS tool [110] and it exhibits time-of-day effect. We cannot provide a comparison with the optimal solution as the CPLEX program is not able to solve the problem for this topology. It failed to fit the optimization model in its memory even though the physical machine had 1TB of memory. The program crashes after the total memory usage reaches around 300 GB. We observed similar behavior when experimenting with high traffic volumes. CPLEX is not able to solve the problem for the Internet2 topology when traffic is increased to utilize the network by more than 40%. We tuned different parameters (*e.g.*, solving the dual problem, storing branch and bound tree data on disk, and reducing the number of threads) of the CPLEX solver according to the guidelines provided by IBM[2], but could not solve the problem. We plan to investigate this issue further in the future. Still, the heuristic solution is able to solve the same problem in less than 3 seconds.

The transit and energy cost for the AS-3967 topology is reported in Figure 3.11. The transit cost is two orders of magnitude higher than the energy cost, which is expected for a larger network with large amount of traffic. From Figure 3.6(c) and Figure 3.11, we can see that our dynamic VNF orchestration approach adapts nicely with the changing traffic conditions. It can dynamically scale-up or scale-down the number of active VNFs (demonstrated by the rise and fall of the energy cost). It can also adapt the location of the VNFs according to the variation in the traffic volume.

The results for system resource utilization and topological properties for middlebox lo-

[2]http://www-01.ibm.com/support/docview.wss?uid=swg21399933



(a) Mean server utilization

(b) Per server utilization

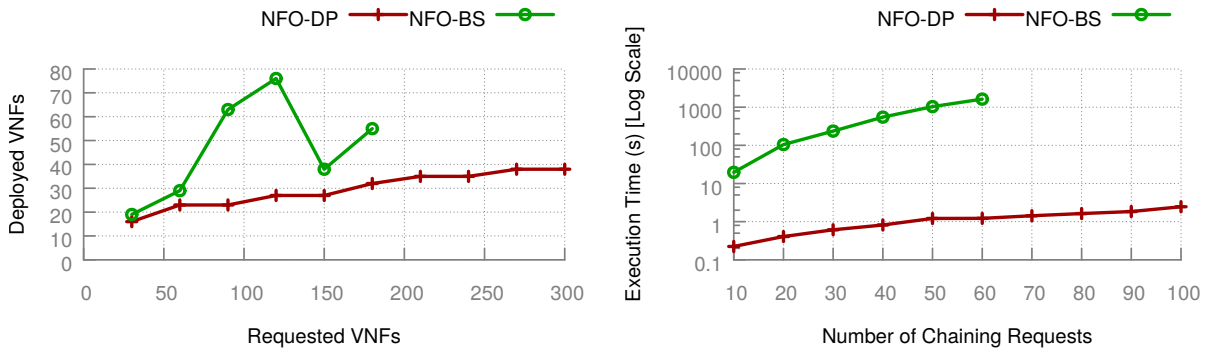
(c) Distance to middlebox

(d) Path stretch

Figure 3.12: Results for Rocketfuel topology (AS-3967)

cations are shown in Figure 3.12. From Figure 3.12(a) we can see that the mean utilization and number of active servers vary with fluctuation in traffic volume. The mean utilization of the servers is around 80%, but there is a small number of servers that are underutilized (Figure 3.12(b)). The CDF of percentage of middleboxes deployed within k-hop distance from the ingress switch is reported in Figure 3.12(c). More than 90% middleboxes are deployed within 5 hops, which is quite reasonable for a network with 79 switches and 147 links. Similar results are obtained for the egress case as shown in the same figure. Finally, the path stretch is shown in Figure 3.12(d). We can observe that 20% traffic passes through the shortest path even after going though the VNF sequence. So, in 20% of the cases VNFs are provisioned on the shortest path between the ingress and egress switches that the traffic is passing through.

3.6.5 Performance Comparison with Previous Work



(a) VNFs deployed

(b) Execution time

Figure 3.13: Performance comparison

We demonstrate the effectiveness of our proposed heuristic (NFO-DP) over prior work by comparing with a very recent and relevant proposal. We implemented the binary search based heuristic proposed in [100] (NFO-BS). We adjusted the heuristic parameters according to the provided guideline in [100]. We experimented with a moderate size ISP network topology with 79 nodes and 147 links (AS3967 from RocketFuel topologies [109]). We varied the number of VNF chaining requests from 10 to 100 and measured the execution time along with the number of deployed VNFs. The results are reported in Figure 3.13. NFO-BS could not find a feasible solution for more than 60 traffic requests within a time

limit of 24 hours. Moreover, the solution quality is not consistent, as shown by the irregular line in Figure 3.13. Our findings show that on similar problem instances NFO-DP outperforms NFO-BS in both solution quality and execution time.

3.6.6 Scalability of Heuristic

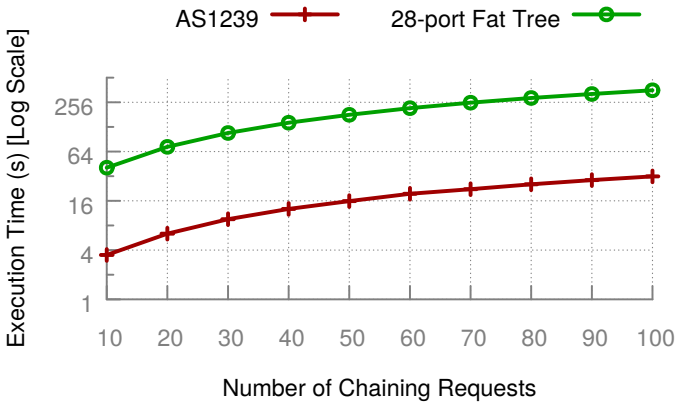
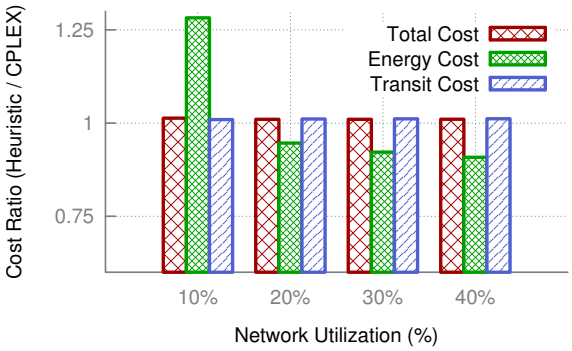


Figure 3.14: Scalability of heuristic

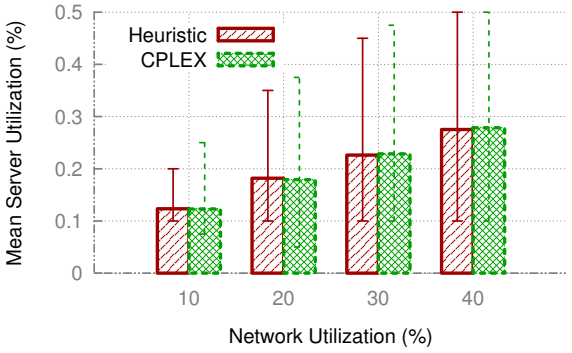
In this scenario, we test the scalability of our proposed heuristic by running it on larger network topologies and report the execution time. For larger network topologies, we used a 28-port fat tree [114] with around 1000 nodes and $10K$ links as a data center network and an ISP network topology with 315 nodes and 972 links (AS1239 from RocketFuel ISP topologies [109]). For each of these topologies we varied the number of VNF chaining requests from 10 to 100 and reported the execution time. Figure 3.14 shows the results of this experiment. As we can see, even for a very large data center network, our proposed heuristic could embed 100 requests in under 6 minutes. It is worth mentioning that the heuristic proposed in [100] could embed only less than 30 VNF chaining requests withing 6 minutes on a much smaller network.

3.6.7 Effect of High Traffic Volume

Now, we show the impact of higher traffic volume on our solution. We perform this experiment by increasing the original traffic by 10% to 40% (in increments of 10%) for the Internet2 topology (Figure 3.15). We observed a linear relationship between OPEX



(a) Cost ratio



(b) Server Utilization

Figure 3.15: Cost ratio (heuristic / CPLEX) with varying load

and network utilization for both of our solutions. The cost also grows almost at the same rate for both CPLEX and heuristic as evident from Figure 3.15(a). The heuristic is able to follow the optimal solution very closely. Although it might seem a bit unintuitive by looking at the ratio of the individual cost components, this occurs as the transit cost is two orders of magnitude larger than the energy cost.

The server utilization increases sub-linearly with increasing network load (Figure 3.15(b)). The number of used servers remains the same for different network loads, but more cores were used since more VNFs were deployed. The larger error bar for CPLEX indicates the deployment of more VNFs, which increases the energy cost. However, more VNFs eventually decreases the transit cost, which is the major contributor to OPEX in this case.

3.7 Related Work

The initial drive for NFV was from several telecommunication operators back in 2013 [44]. The motivation behind NFV is to break the barrier of proprietary hardwares and have more flexibility in the network in terms of service placement, introducing new services, and vendor independence. To this date, research efforts have been made in different aspects of NFV. In this section, we first discuss about state-of-the-art NFV management and orchestration proposals (Section 3.7.1), then we describe some placement algorithms for VNFs and VNF chains (Section 3.7.2), followed by some enabling technologies for NFV (Section 3.7.3).

3.7.1 Management and Orchestration of Network Functions

Some of the early works on managing VNFs, propose to outsource them to a cloud service [14, 115]. Such outsourcing is motivated in the literature by studying experiences of different network operators. [14, 115] show how the management complexities arising in today's enterprise networks can be mitigated through outsourcing.

A more formal management approach towards NFV is taken by projects such as Stratos [116], and OpenNF [61]. Stratos proposes an architecture for orchestrating VNFs outsourced to a remote cloud by taking care of traffic engineering and horizontal scaling. On the other hand, OpenNF proposes a converged control plane for VNFs and network forwarding plane by extending the centralized SDN paradigm. Our proposed orchestration algorithms for VNF chain placement can be used by such management systems.

Some recent works on managing VNFs focus on traffic engineering issues such as steering the traffic through some existing sequence of VNFs [117, 37]. This problem becomes more challenging when some VNF modifies the packet headers, thus changing the traffic signature. Authors in [117, 37] propose tagging based mechanisms to identify a traffic during its lifetime and also to keep track of the visited sequence of VNFs. These works are complementary to ours as we focus on determining the placement and traffic routing paths for VNF chains, while the tagging based approaches can be utilized to deploy the VNF chain in an SDN network.

3.7.2 VNF and VNF Chain Placement

Authors in [118] proposed a grammar for specifying VNF chains and then provided a mathematical formulation for VNF chain placement. Their formulation is quadratic and

does not allow VNF sharing between multiple tenants. In contrast, we provide a linear formulation and allow for VNF sharing. In [119], the authors provided an LP-relaxation based approach for finding inter-data center VNF chain placement. However, due to LP-relaxation, their solution violates physical resource capacities by a factor of at most 16. Our solutions do not have such issue, and we provide extensive simulations to show that our proposed heuristic achieves near-optimal performance within a second. A genetic algorithm for VNF chain placement is proposed in [120], but it does not address the issue of dynamically adjusting the placement of VNFs to balance between network operating cost and performance. An orchestration architecture for automated VNF placement is proposed in [121], but the authors do not provide any concrete algorithms for orchestration. Our proposed solutions can be used by such systems to determine the placement locations.

The VNF placement problem is also addressed in [122, 123, 124, 125], where the authors formulated the problem as an ILP or MILP and proposed heuristic algorithms. Pham *et al.* provided a sampling-based Markov approximation solution for this problem [126], while Ma *et al.* formulated this as a graph optimization problem [127]. However, these works solve the problem in a two-step process. In the first step the VNF locations are determined, and in the second step, routing paths between these locations are selected. In contrast, our proposed solution determines the VNF placement and routing paths in a single shot.

3.7.3 Enabling Technologies for NFV

In recent years, a number of research efforts have been targeted to achieve near line speed network I/O throughput with commodity servers [128, 129]. Apart from accelerating the packets along the network I/O stack, more recent works have proposed changes to virtualization technologies to support deployment of modular VNFs on lightweight VMs [45]. Hundreds of these VMs can be instantiated on a single physical machine within milliseconds. CoMb [130] and xOMB [131] propose an extensible and consolidated framework for incrementally developing scalable middleboxes by leveraging the idea of reusable network processing pipelines. These works are orthogonal to our work. They focus on developing the technologies for scalable VNFs, while we focus on optimization algorithms for VNF chain placement.

3.8 Conclusion

Virtualized network functions provide a flexible way to deploy, operate and orchestrate network services with much less capital and operational expenditures. Software middleboxes

(*e.g.*, ClickOS) are rapidly catching up with hardware middlebox performance. Network operators are already opting for NFV based solutions. We believe that our model for dynamic VNF orchestration will have significant impact on middlebox management in the future. Our model can be used to determine the optimal number of VNFs and to place them at the optimal locations to optimize network operational cost and resource utilization. Our trace driven simulations on the Internet2 research network demonstrate that network OPEX can be reduced by a factor of 4 over hardware middleboxes through proper VNF orchestration. In this chapter, we presented two solutions to the VNF orchestration problem: CPLEX based optimal solution for small networks and a heuristic for larger ones. We found that the heuristic produces solutions that are within 1.3 times of the optimal solution, yet the execution-time is about 65 to 3500 times faster than that of the CPLEX solution.

Chapter 4

Energy Smart Service Function Chain Orchestration

4.1 Introduction

The rapid proliferation of bandwidth intensive and latency sensitive applications like on-demand video streaming, augmented/virtual reality, Internet-of-things, *etc.* is pushing the Information and Communication Technology (ICT) industry to expand continuously. For example, AT&T, one of the leading telecommunication service providers in the US, reportedly experienced 100,000 percent increase in traffic over a period of eight years (2008 to 2016) [132]. Such large-scale expansion is also increasing the energy requirement of ICT infrastructure and contributing to the global carbon emissions [49]. Statistics from 2013 show that telecommunication infrastructure alone accounted for 47% of the total energy consumed by the ICT sector [48]. When translated into carbon footprint, this sector produced 2.5% of global carbon footprint, and this is projected to double by 2020 [50]. Governments are trying to minimize the environmental impact of carbon emission by introducing regulations and taxes, driving companies to use renewable energy (*i.e.*, green energy) [51, 52, 53]. However, renewable energy is still not cost-effective compared to traditional sources of energy (*i.e.*,brown energy), and their availability varies significantly across time and geographic locations. Therefore, it is a challenge for telecommunication companies to comply with regulations and minimize carbon footprint [55, 56, 57] without significantly increasing their operational cost.

In order to address the challenge mentioned above, telecommunication network operators can employ several possible approaches such as reduce electricity consumption by

optimizing resource usage, buy energy from green(er) energy vendors, install on-site renewable energy sources (*e.g.*, wind turbines or solar panels), *etc.* For example, Google achieved zero carbon emission for its data centers in 2017, by combining on-site renewable sources with energy bought from third-party renewable energy vendors [133]. In this paper, we particularly focus on reducing carbon emission of a telecommunication network by increasing resource efficiency through the following two techniques: (i) migration of network services between different geographic locations to maximize the utilization of on-site renewable energy sources, and (ii) intelligent topology aware placement and consolidation of network services to save energy by opportunistically switching off unused equipments. To this end, we propose an **E**nergy **S**mart **S**ervice **O**rchestrator (ESSO) that employs these techniques to reduce the overall carbon footprint of a telecommunication network.

A major obstacle in implementing ESSO, is the telecommunication operators' reliance on proprietary and vertically integrated hardware *middleboxes* for realizing different network services [14]. Telecommunication operators typically deploy middleboxes such as firewall, load-balancer, WAN optimizer, intrusion detection and prevention systems in Telecommunication Central Offices (COs) [132] and Point-of-Presence (PoP) locations. In recent year, these locations have evolved into mini-data centers that host small to moderate number of servers and network equipments. COs and PoPs are geographically distributed and located in cities and metropolitan areas to provide broadband and wireless Internet services to both residential and commercial customers. A telecommunication service provider's transport network is typically organized into edge, metro, and core regions [134] as shown in Figure 4.1. Typically, there are 10k to 100k access locations within a region, which host resources very close to the subscriber. The access locations are aggregated at the Tier-1 CO or PoP locations, which are further aggregated at the Tier-2 metro CO locations. Finally, the core transport network connects all these locations to the core data center of the network.

Depending on the type of traffic, network flows at a CO are processed through different sequences of middleboxes or Service Function Chains (SFCs) [135] to ensure fast, reliable, and secure access to the Internet [13]. Despite the telecommunication operators' extensive reliance on middleboxes, their closed and vertically-integrated nature pose a number of operational challenges [14]. Hardware middleboxes are attached to fixed locations in the network and offer little or no programmability and dynamicity [14]. As a result, it becomes a daunting task to dynamically migrate SFCs across COs to leverage on-site renewable energy or consolidate SFCs withing the same location to turn-off underutilized equipment.

Recently, telecommunication networks are going through a transformation known as *network softwarization* [136]. It provides the necessary flexibility and agility to dynamically control network elements and traffic flows that are required to build ESSO. Software

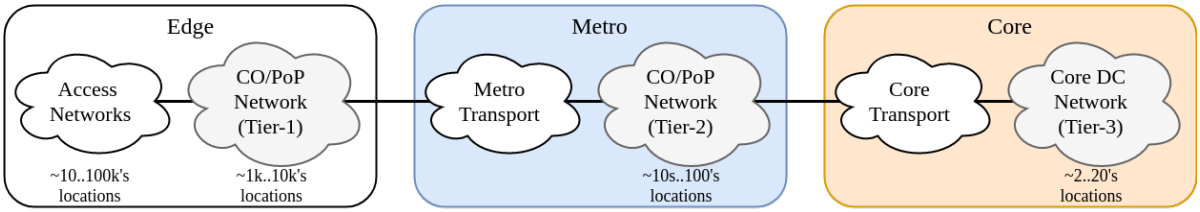


Figure 4.1: Tiered network structure

Defined Networking (SDN) [137], and Network Functions Virtualization (NFV) [11] are two cornerstones of network softwarization. On one hand, SDN decouples the network's control and data planes and implements the control plane as a logically centralized software; facilitating network programmability. NFV, on the other hand, moves the data plane from proprietary hardware middleboxes to software in the form of Virtual Network Functions (VNFs) running on commodity servers. SDN and NFV simplify network infrastructure by permitting the utilization of inexpensive commodity off-the-shelf (COTS) hardware for both compute and network; bringing benefits such as economies-of-scale, agility, and flexibility of cloud computing to the COs and PoPs of telecommunication networks. In the rest of the chapter, we refer to locations like COs, PoPs, and core data centers, which host servers to deploy VNFs, as NFV Point-of-Delivery or NFV-PoDs, as all these locations can be used to deploy and deliver a network service.

With SDN and NFV, VNFs are no longer restricted to fixed locations; they can be provisioned on any compute server within the network. Moreover, coupled with an SDN controller; VNFs and their associated traffic flows can be migrated within a short timeframe to a different compute server [61, 62, 63] within or across NFV-PoDs. The flexibility provided by SDN and NFV make it possible to compose and reconfigure network services on the fly. ESSO builds on SDN and NFV and reduces the overall energy consumption and carbon footprint by (i) opportunistically utilizing more resources at locations with surplus renewable energy while minimizing consumption at locations where brown energy is the only option, and (ii) taking VNF placement and consolidation decisions in a manner that allows switches, switch ports, and servers to be put into low-power consumption state to reduce the overall power consumption. Specifically, this work consists of the following contributions:

- An ILP formulation for the SFC orchestration problem across multiple NFV-PoDs that considers the availability of renewable energy.
- An architectural design for an SFC orchestrator that enables efficient resource and

energy utilization for a geographically distributed infrastructure.

- A set of heuristic algorithms for taking SFC embedding, consolidation, and migration decisions across the NFV-PoDs of a telecommunication network.

The rest of the chapter is organized as follows: Section 4.2 provides the required background to delineate the concepts applied in this work. The architecture of ESSO is presented in Section 4.3. Next, we provide a mathematical formulation for the network service orchestration problem in Section 4.4. Section 4.5 presents a set of heuristic algorithms for the placement, consolidation, and migration of network services based on fluctuations in the availability of renewable energy. After that, we present the results obtained from performance evaluation of the presented algorithms in Section 4.6. We discuss related work in Section 4.7, and finally, conclude with some future research directions in Section 4.8.

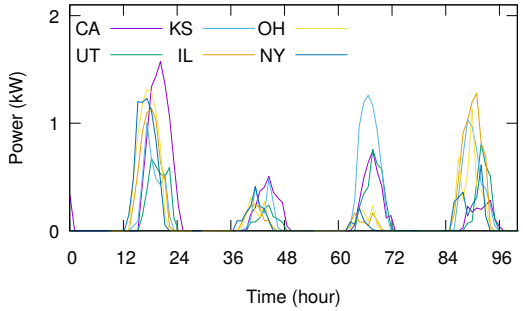
4.2 Background

4.2.1 Fluctuation in Availability of Renewable Energy

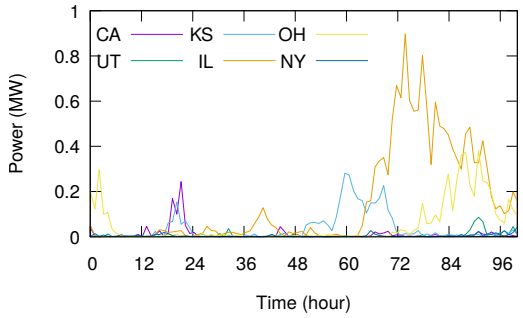
The availability of renewable energy sources such as solar and wind significantly fluctuates across different locations and time. They are very intermittent in nature and fluctuate with both time and weather conditions even at the same location. Using the data collected by the U.S. Climate Reference Network and Regional Climate Reference Network [60] for June 2017, estimated energy generation potential from solar and wind for six US cities in different states are shown in Figure 4.2(a) and Figure 4.2(b), respectively. The solar energy is estimated for a single $4m^2$ PV panel, and the wind energy is estimated for a 250 feet diameter wind turbine [138]. As we can see from the figure, solar energy follows a diurnal pattern. However, wind energy does not show any particular pattern. Telecommunication operators have initiated different renewable energy projects to reduce their carbon footprint. For example, in 2014, AT&T expanded its solar energy capacity to 3 MW in California and New York [139]. AT&T is also planning to achieve 60 percent reduction in brown energy consumption by 2020 and ten times reduction in carbon footprint by 2025 [140].

4.2.2 Variation in Different Electricity Grids' Carbon Footprint

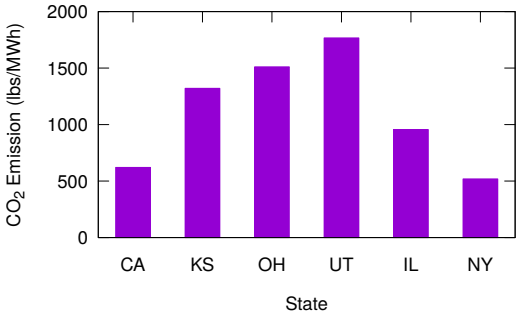
Electricity grids in different areas use different technologies and hence the carbon footprint per MWh of energy is not the same across different locations. Figure 4.2(c) shows the av-



(a) Solar energy



(b) Wind energy



(c) Grid carbon footprint

Figure 4.2: Renewable energy and grid carbon footprint data (June 2017)

erage carbon footprint (lbs/MWh) for the same six states reported in the previous section. ESSO considers the variation in carbon footprint between different locations while making placement decisions. We collected data on the carbon footprint of local electricity grids for various states in the U.S. [141] and used them during performance evaluation.

4.2.3 Energy Consumption States of Networking Equipment

A network switch usually consumes the major portion of power when it is just turned on [142]. Ideally, consumed power should increase with the amount of traffic processed by the switch; however, in reality, it is not the case. For example, the maximum wattage of the Dell PowerConnect 8024F switch is 160 Watts, and it consumes around 110 Watts without any traffic load [143]. However, the switch consumes only 10 Watts when put into sleep mode [59]. Switch ports consume power depending on the configured maximum link rate. For the Dell PowerConnect 8024F switch, a port in 1 Gbps and 10 Gbps link rate settings consumes 1.2 and 4.3 Watts, respectively [59]. Switch ports consume the same amount of power regardless of their utilization. A number of works have focused on the energy efficiency of SDN-enabled switches as well [144, 58]. According to Vu *et al.* [58], for an OpenFlow-enabled switch, sleep mode for the switch ports can save around 9.8% of total power, and sleep mode for the entire switch can save 60% of total power.

4.2.4 Central Office Re-architected as Data Center (CORD)

The CORD [145] project combines features from cloud, SDN, and NFV to re-architecture telecommunication COs. It proposes to organize commodity servers, switches, and GPON access ports in a leaf-spine fabric similar to a data center. CORD brings the economies of scale and flexibility of data centers to telecommunication COs. Additional details about the architecture of CORD can be found in [146]. In this work, we assume that the COs and PoPs have been re-architected by using a concept similar to CORD, and refer to them as NFV-PoDs. We also assume that VNFs can be deployed on the compute servers and an SDN controller dynamically routes flows within and across NFV-PoDs.

4.2.5 SFC Migration and Virtualization Technology

A network function can be softwarized in many different ways. For example, a firewall, can be deployed on a Virtual Machine (VM) running on-top of a hypervisor such as Xen or KVM. We can also deploy it using lightweight OS containers such as LXC, Docker, or

Rocket. Furthermore, VNFs can be deployed in a holistically different environment such as those considered by OpenNF [61], Split/Merge [63], and Stateless [147], where the internal state of the VNF is decoupled from its processing logic. In most cases the state is stored in a networked, fast, and reliable distributed storage system. Depending on the technology used to deploy a VNF, migration times can vary widely. If a VNF is deployed in a VM, then the traditional migration of VMs with moderate amount of downtime applies. In case of containers, the downtime is significantly less than that of a VM migration. Moreover, in case of solutions like OpenNF or Stateless, the migration time is negligible. In [62], authors demonstrate how a moderately loaded state-decoupled VNF (like OpenNF) can be migrated with negligible service disruption.

4.3 System Architecture

4.3.1 Assumptions

We make the following assumptions regarding the underlying infrastructure and energy usage policy: Without loss of generality, we assume a telecommunication network architecture similar to the one depicted in Figure 4.3. NFV-PoDs are geographically distributed and connected through access, metro, and core transport networks. They serve residential, commercial, and mobile customers by providing last-mile and wireless connectivity. Each NFV-PoD contains a set of compute and networking resources. Depending on the size of customer base supported by a NFV-PoD, the volume of servers can vary from a few servers to multi-rack deployments. In this work, we assume that each NFV-PoD hosts a leaf-spine or fat-tree like network topology along with multiple servers or racks of servers for hosting VNFs.

The number of NFV-PoDs in a single city or metropolitan area depends on population density and can range from 10s to 100s [148]. Each NFV-PoD may or may not have on-site renewable energy sources (*e.g.*, solar, wind, *etc.*) and resorts to the traditional electricity grid when the amount of renewable energy is insufficient. Renewable energy sources are considered carbon free, while the energy drawn from the grid (brown energy) generates a certain amount of carbon depending on the particular electricity generation technique used by local energy grid.

ESSO keeps track of the following metrics at each NFV-PoD: (i) amount of available compute and networking resources, (ii) lease duration of allocated resources for currently deployed SFCs, (iii) availability of renewable energy, and (iv) carbon footprint of brown

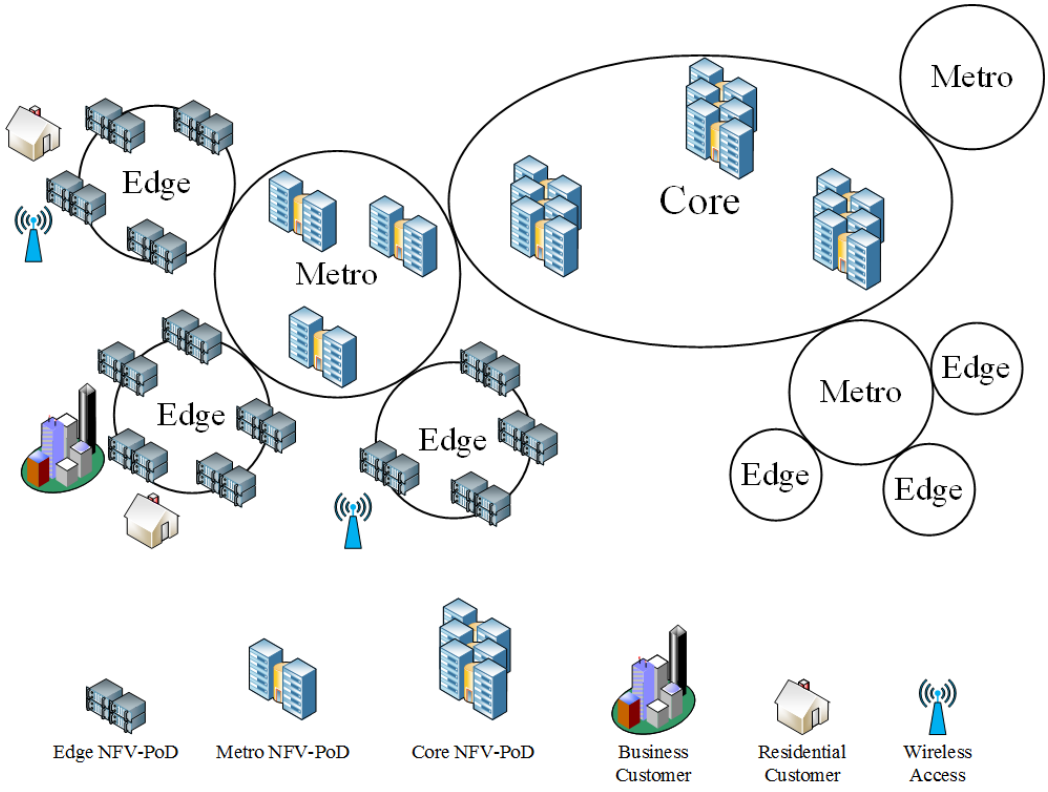


Figure 4.3: Multi-tiered NFV-PoD network

energy. It also tracks the amount of available bandwidth on the inter-NFV-PoD backbone links. ESSO receives SFC requests and based on the above mentioned metrics, takes orchestration decisions with the high-level objective to reduce the overall carbon footprint of the network. Service chains are migrated based on the (i) fluctuation in availability of renewable energy and (ii) expiry of previously deployed service chains.

4.3.2 ESSO Architecture

Our proposed orchestration architecture has two components: a Global Orchestrator (G-Or) and one Local Orchestrator (L-Or) for each NFV-PoD. G-Or monitors the overall network infrastructure, receives service chaining requests, allocates resources across multiple NFV-PoDs to deploy service chains, and migrates service chains within or across NFV-PoDs. Each NFV-PoD has a Local Orchestrator (L-Or) that monitors and collects information about the intra-NFV-PoD network links, switches, and server resources and sends the data to the G-Or. The G-Or accumulates data received from the L-Ors and creates a global view of the entire infrastructure. The L-Ors collect data regarding the amount of renewable energy and track the usage of both renewable and brown energy, which are used to measure the total carbon emissions. Data provided by the L-Ors allow the G-Or to optimize resource allocation and migration decisions. In the following, we describe the architecture of L-Or and G-Or in detail.

L-Or Architecture

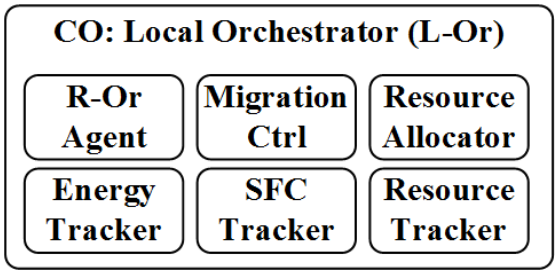


Figure 4.4: Components of local orchestrator

The components of an L-Or are shown in Figure 4.4. There are six components: (i) Resource Tracker, (ii) SFC Tracker, (iii) Energy Tracker, (iv) Resource Allocator, (v) Migration Controller, and (iv) R-Or Agent. As their names suggest, the first three components keep track of local resources (*e.g.*, intra-NFV-PoD servers, switches, and links),

currently embedded SFCs, and energy usage, respectively. The resource allocator receives SFC embedding requests from the G-Or and determines the mapping of VNFs on physical servers and switches based on local resource availability. Then, it instantiates the required virtual machines or containers to deploy the VNFs and also creates the virtual links between them to instantiate the SFC. The migration controller is responsible for migrating a VNF from one server to another, along with its incident virtual links. The source and destination physical servers involved in the migration can reside either in the same NFV-PoD or different NFV-PoDs. The R-Or Agent facilitates the communication between an L-Or and the G-Or.

G-Or Architecture

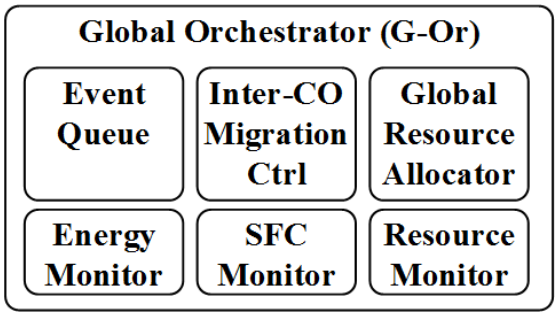


Figure 4.5: Components of global orchestrator

The components of the G-Or are shown in Figure 4.5. The Resource, SFC, and Energy Monitors collect data from the L-Ors used during the embedding process. The Global Resource Allocator receives an SFC request and then decides the placement of VNFs from the SFC. It considers the availability of renewable energy sources, the amount of free compute and networking resources at different NFV-PoDs, and makes embedding decisions. We assume that each SFC request has an expiry time; after which the SFC is removed from the system. The Inter-NFV-PoD Migration Controller keeps track of SFC expiry events and changes in the availability of renewable energy at different NFV-PoDs. When an existing SFC expires and leaves the system, the freed resources can be used to consolidate existing SFCs to reduce power consumption. When a renewable energy source starts generating electricity, *e.g.*, there is a strong wind or during morning hours when the sun rises; the migration controller makes migration decisions to take advantage of surplus renewable energy. The migration controller decides a VNF migration only when the migration reduces the total carbon footprint of the SFC.

4.4 Problem Formulation

In this section, we first formally define the "Multi-Location SFC Orchestration" or *MLSO* problem and then provide a mathematical formulation to construct an Integer Linear Programming (ILP) model for it.

4.4.1 Multi-Location SFC Orchestration (MLSO) Problem

In the MLSO problem, the resource capacities of compute and networking resources, carbon footprint per unit of brown energy, availability of renewable energy for each NFV-PoD, and the link bandwidth of the inter-NFV-PoD backbone network are provided as input. A stream of SFC requests is also provided for which the ILP solution will determine the optimal embedding. The amount of available renewable energy varies over time. The carbon footprint of brown energy is different at different locations. A solution to the MLSO problem must utilize these facts to reduce the overall carbon footprint of the telecommunication network.

Each SFC request consists of an ingress NFV-PoD, an egress NFV-PoD, and a sequence of Network Function (NF) types and flavors that the traffic needs to pass through. Types distinguish NFs into categories like firewall, NAT, IPS, proxy, *etc.* and flavors represent VNF variations in terms of resource requirements. For example, a firewall can have two flavors: (i) one uses 2 cpu cores and can process traffic at a maximum rate of 500Mbps and (ii) the other one uses 4 cpu cores and can process at a maximum rate of 900Mbps.

We assume that embedding, migration, or consolidation decisions are made at particular time-instances. These time-instances are identified by the occurrence of one or more of the following events: (i) arrival of a new SFC request, (ii) departure of one or more previously embedded SFC request(s), and (iii) changes in the amount of available renewable energy. When a new SFC request arrives, it needs to be embedded. After the departure of one or more previously embedded SFC request(s), resources at the corresponding NFV-PoDs can be consolidated. If the availability of renewable energy changes in a subsequent time-instance then existing (*i.e.*,already embedded) VNFs can be migrated to a different NFV-PoD. A VNF is migrated only if the cost of migration is less than the savings achieved in terms of carbon footprint. The ILP makes embedding, migration, and/or consolidation decisions based on the set of events for each time-instance. The objective of the ILP is to minimize carbon footprint by determining the initial placement of new SFC requests, subsequent consolidation and migration of existing VNFs based on the availability of resources and renewable energy, while guaranteeing that the total latency of any SFC does

not violate the maximum allowed delay constraint.

The MLSO problem is $\mathcal{NP}$-Hard, because for each time-instance of the MLSO problem, we actually need to solve the general SFC orchestration problem [149] for the newly arriving SFCs, which is known to be $\mathcal{NP}$-Hard [43]. The general SFC orchestration problem can be reduced to the MLSO problem by considering one time-instance and assuming all SFC requests as new requests that need to be embedded. In this setting, a solution to the MLSO problem will also be a solution to the general SFC orchestration problem, and hence the MLSO problem is $\mathcal{NP}$-Hard.

4.4.2 Input Representation

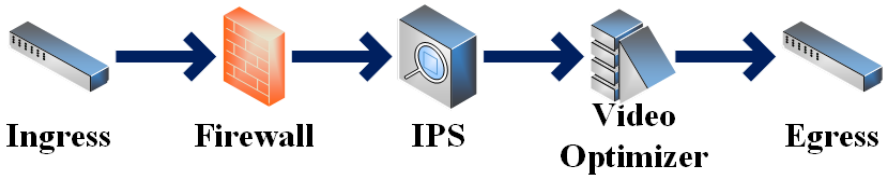


Figure 4.6: A Service Function Chain (SFC)

An SFC request is composed of one of more VNFs belonging to various types. Figure 4.6 demonstrates an SFC consisting of the chain `Firewall` $\rightarrow$ `IPS` $\rightarrow$ `Video Optimizer`. The nodes at the beginning and the end represent the ingress and egress NFV-PoDs, respectively. For each VNF type $p \in P$, we have a set of flavors F_p, and the function $\phi(f)$ (for flavor $f \in F_p$) returns the type (p) of a VNF. The resource requirements of a particular VNF flavor f, is represented by $\varphi_f^r \in \mathbb{R}^+$, $\forall\ r \in R$, where R represents resource types like CPU, memory, disk-space, *etc.* We also assume that a VNF of flavor f introduces a traffic processing delay of δ_f.

Let, $\mathcal{I}_n^t$ and $\mathcal{I}_p^t$ represent the set of new (incoming) and pre-existing SFC requests in the system at time-instance $t \in \mathcal{T}$, respectively. Here, the set $\mathcal{T}$ represents all possible time-instances considered for the ILP. Each SFC request has an associated active duration or lifetime, after which it leaves the system and is removed from the set $\mathcal{I}_p^t$. Therefore, at each time-instance, the ILP needs to take migration and/or consolidation decisions for the SFCs in $\mathcal{I}_p^t$ and at the same time determine the embedding of the SFCs in $\mathcal{I}_n^t$. We also define the set $\mathcal{I}^t = \mathcal{I}_n^t \cup \mathcal{I}_p^t$ to represent both new and pre-existing SFC requests at time-instance t.

We define the following binary variables to establish the active duration, arrival, and departure events for an SFC:

$$\ddot{\nu}_i^t = \begin{cases} 1 & \text{if SFC } i \in \mathcal{I} \text{ arrives at time-instance } t \in \mathcal{T}, \\ 0 & \text{otherwise.} \end{cases}$$

$$\ddot{a}_i^t = \begin{cases} 1 & \text{if SFC } i \in \mathcal{I} \text{ is active at time-instance } t \in \mathcal{T}, \\ 0 & \text{otherwise.} \end{cases}$$

$$\ddot{d}_i^t = \begin{cases} 1 & \text{if SFC } i \in \mathcal{I} \text{ departs at time-instance } t \in \mathcal{T}, \\ 0 & \text{otherwise.} \end{cases}$$

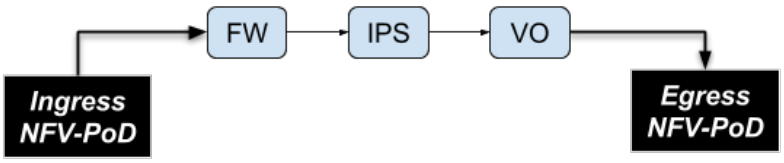


Figure 4.7: Graph representation of an SFC

An SFC request, $i \in \mathcal{I}^t$, is represented as a directed acyclic graph $G_i = (N_i, L_i)$ as shown in Figure 4.7. Here, N_i represents the set of nodes in the SFC. There are two types of nodes; placeholder nodes for designating the ingress and egress NFV-PoDs of the chain and VNF nodes. The embedding location of the placeholder nodes are predetermined as they need to be embedded at particular NFV-PoDs. The embedding of VNF nodes need to be determined. Each VNF node $n \in N_i$, is of a particular NF flavor denoted by the function $f(n)$. The flavor of an NF determines its resource requirements. Each link $l \in L_i$ has a bandwidth requirement β_l. The maximum tolerable delay for the entire chain is δ_i. We also define two functions $s(.)$ and $d(.)$ to denote the source and destination nodes of each link $l \in L_i$.

4.4.3 Physical Infrastructure Representation

The physical infrastructure is represented by a graph $\bar{G} = (\bar{N} \cup \bar{S}, \bar{L})$, where $\bar{N}$ and $\bar{S}$ represent the servers and switches of all NFV-PoDs, respectively. $\bar{L}$ represents the inter-

and intra-NFV-PoD links of the telecommunication network. The set C, represents the NFV-PoDs in the network and an individual NFV-PoD is denoted by c. We define e_r^{tc} as the amount of available renewable energy at NFV-PoD c during time-instance t. The amount of carbon per watt of brown power generated by the local electricity grid is represented by ζ_c and the Power Usage Effectiveness (PUE) of a NFV-PoD is represented by μ_c.

Let, R denote the set of resources (CPU, memory, disk, *etc.*) offered by a server. The resource capacity of server $\bar{n} \in \bar{N}$ is denoted by $\kappa_{\bar{n}}^r \in \mathbb{R}^+,\ \forall\ r \in R$. The ports of a switch $\bar{s} \in \bar{S}$ is represented by the set $\bar{\Pi}_{\bar{s}}$ and individual ports are represented by $\bar{\pi} \in \bar{\Pi}_{\bar{s}}$. The operational mode of a port is denoted by the function $m(\bar{\pi})$. $e_{m(\bar{\pi})}$ denotes a port's energy consumption in a particular operational mode. The bandwidth capacity and propagation delay of a physical link $\bar{l} \in \bar{L}$ is represented by $\beta_{\bar{l}} \in \mathbb{R}^+$ and $\delta_{\bar{l}} \in \mathbb{R}^+$, respectively. Energy consumed per unit bandwidth is represented by ξ^β.

Here, we assume a linear power model for server power consumption [150]. So, $e_{\bar{n}}^b$ and $e_{\bar{n}}^m$ represent the base and max power consumption of a server, respectively. The base power is consumed when the server is active but there is no workload. Next, $e_{\bar{n}}^s$ represents the power consumption when the server is in sleep mode. We define the following binary variable to denote whether a server is active or not. This variable is utilized to measure the total power consumption.

$$\ddot{a}_{\bar{n}}^t = \begin{cases} 1 & \text{if } \bar{n} \in \bar{N} \text{ is active between } [t, t+1), \\ 0 & \text{otherwise.} \end{cases}$$

Next, we define an additional binary variable to denote whether a switch port is active or not. This variable is utilized to measure the total energy consumption of a switch.

$$\ddot{a}_{\bar{\pi}\bar{s}}^t = \begin{cases} 1 & \text{if port } \bar{\pi} \text{ is active between } [t, t+1), \\ 0 & \text{otherwise.} \end{cases}$$

A switch is considered to be in the active mode if at least one of its ports is active, otherwise it is in the sleep mode. The base energy of a switch $\bar{s}$ in active mode is represented by $e_{\bar{s}}^b$ and the energy consumption of a switch in sleep mode is denoted by $e_{\bar{s}}^s$. The total energy consumption of an active switch depends on the number of active ports and their operational modes, as discussed in Section 4.2.3. If all ports of a switch are inactive, then the switch is considered to be in the sleep mode and consumes substantially less energy. The following binary variable denotes whether a switch is in active mode or in sleep mode:

$$\ddot{a}^t_{\bar{s}} = \begin{cases} 1 & \text{if switch } \bar{s} \text{ is active between } [t, t+1), \\ 0 & \text{otherwise.} \end{cases}$$

All possible paths between the servers and switches in $\bar{N} \cup \bar{S}$ are represented by $\bar{P}$, and the following variable denotes whether a physical link $\bar{l} \in \bar{L}$ belongs to a path:

$$\psi_{\bar{l}\bar{p}} = \begin{cases} 1 & \text{if link } \bar{l} \in \bar{L} \text{ belongs to path } \bar{p} \in \bar{P}, \\ 0 & \text{otherwise.} \end{cases}$$

The following binary variable expresses the connectivity between physical paths and switch ports:

$$w_{\bar{\pi}\bar{s}\bar{p}} = \begin{cases} 1 & \text{if port } \bar{\pi} \text{ of switch } \bar{s} \text{ is on physical path } \bar{p} \in \bar{P}, \\ 0 & \text{otherwise.} \end{cases}$$

Next, we define $\bar{N}^c$, $\bar{S}^c$, $\bar{L}^c$, and $\bar{P}^c$ as the set of servers, switches, physical-links, and physical-paths belonging to NFV-PoD, $c \in \mathcal{C}$. We define the function $\bar{s}(.)$ and $\bar{d}(.)$ to denote the source and destination nodes of a physical link and path. The function $\nu(.)$ maps a server or link to it's corresponding NFV-PoD. We also define the function $\mathcal{B}(\bar{P})$ to denote the backbone links between NFV-PoDs on the physical paths. Next, $\beta_{\bar{p}}$ denotes the available bandwidth of the path $\bar{p}$ and is equal to the minimum bandwidth capacity over all physical links in a path: $\beta_{\bar{p}} = \min_{\bar{l} \in \bar{p}} \beta_{\bar{l}}$.

There can be certain hardware requirements (*e.g.*, hardware-accelerated encryption for Deep Packet Inspection (DPI)) that may prevent a server from running a particular type of VNF. Furthermore, the network operator may have preferences regarding provisioning a particular type of VNF on a particular set of servers, *e.g.*, Firewalls should be deployed close to the network edge. So, we assume that for each VNF type there is a set of servers on which it can be provisioned. The following binary variable represents this relationship:

$$d_{\bar{n}p} = \begin{cases} 1 & \text{if VNF of type } p \in P \text{ can be provisioned on } \bar{n}, \\ 0 & \text{otherwise.} \end{cases}$$

Moreover, each VNF in a chain can have location restrictions expressed by the following binary variable:

$$z_n^{ci} = \begin{cases} 1 & \text{if VNF } n \in N_i \text{ can be provisioned in NFV-PoD } c, \\ 0 & \text{otherwise.} \end{cases}$$

4.4.4 Decision Variables

Now, we define the following decision variables to denote the placement of a VNF and the routing path of a chain on the physical infrastructure:

$$\hat{x}_{n\bar{n}}^{ti} = \begin{cases} 1 & \text{if VNF } n \in N_i \text{ provisioned on } \bar{n} \in \bar{N} \text{ between } [t, t+1), \\ 0 & \text{otherwise.} \end{cases}$$

$$\hat{y}_{l\bar{p}}^{ti} = \begin{cases} 1 & \text{if link } l \in L_i \text{ provisioned on } \bar{p} \in \bar{P} \text{ between } [t, t+1), \\ 0 & \text{otherwise.} \end{cases}$$

4.4.5 ILP Formulation

The above mentioned variables must satisfy the following constraints of the optimization problem:

$$\sum_{\bar{n} \in \bar{N}} \hat{x}_{n\bar{n}}^{ti} = \ddot{a}_i^t, \ \forall\ t \in \mathcal{T}, i \in \mathcal{I}^t, n \in N_i \tag{4.1}$$

$$\sum_{\bar{p} \in \bar{P}} \hat{y}_{l\bar{p}}^{ti} = \ddot{a}_i^t, \ \forall\ t \in \mathcal{T}, i \in \mathcal{I}^t, l \in L_i \tag{4.2}$$

$$\hat{x}_{n\bar{n}}^{ti} \leq d_{\bar{n}\phi(f(n))}, \ \forall\ t \in \mathcal{T}, i \in \mathcal{I}^t, n \in N_i, \bar{n} \in \bar{N} \tag{4.3}$$

$$\hat{x}^{ti}_{n\bar{n}} \leq z^{\nu(\bar{n})i}_{n},\ \forall\ t \in \mathcal{T}, i \in \mathcal{I}^t, n \in N_i, \bar{n} \in \bar{N} \tag{4.4}$$

$$\hat{y}^{ti}_{l\bar{p}} \leq \hat{x}^{ti}_{s(l)\bar{s}(\bar{p})},\ \forall\ t \in \mathcal{T}, i \in \mathcal{I}^t, s(l) \in N_i, \bar{p} \in \bar{P} \tag{4.5}$$

$$\hat{y}^{ti}_{l\bar{p}} \leq \hat{x}^{ti}_{d(l)\bar{d}(\bar{p})},\ \forall\ t \in \mathcal{T}, i \in \mathcal{I}^t, d(l) \in N_i, \bar{p} \in \bar{P} \tag{4.6}$$

$$\sum_{t \in \mathcal{T}} \sum_{i \in \mathcal{I}^t} \sum_{n \in N_i} \hat{x}^{ti}_{n\bar{n}} \times \varphi^{r}_{f(n)} \leq \kappa^{r}_{\bar{n}},\ \forall\ \bar{n} \in \bar{N}, r \in R \tag{4.7}$$

$$\sum_{t \in \mathcal{T}} \sum_{i \in \mathcal{I}^t} \sum_{n \in N_i} \hat{y}^{ti}_{l\bar{p}} \times \beta_{l} \leq \beta_{\bar{p}},\ \forall\ \bar{p} \in \bar{P} \tag{4.8}$$

$$\sum_{l \in L_i} \hat{y}^{ti}_{l\bar{p}} \times \delta_{\bar{p}} + \sum_{n \in N_i} \delta_{f(n)} \leq \delta_{i},\ \forall\ t \in \mathcal{T}, i \in \mathcal{I}^t \tag{4.9}$$

$$\ddot{a}^{t}_{\bar{n}} \leq \sum_{i \in \mathcal{I}^t} \sum_{n \in N_i} \hat{x}^{ti}_{n\bar{n}},\ \forall\ t \in \mathcal{T}, \bar{n} \in \bar{N} \tag{4.10}$$

$$\ddot{a}^{t}_{\bar{n}} \geq \frac{\sum_{i \in \mathcal{I}^t} \sum_{n \in N_i} \hat{x}^{ti}_{n\bar{n}}}{\sum_{i \in \mathcal{I}} |N_i|},\ \forall\ t \in \mathcal{T}, \bar{n} \in \bar{N} \tag{4.11}$$

$$\ddot{a}^{t}_{\bar{s}} \leq \sum_{\bar{\pi} \in \bar{\Pi}} \ddot{a}^{t}_{\bar{\pi}\bar{s}},\ \forall\ t \in \mathcal{T}, \bar{s} \in \bar{S} \tag{4.12}$$

$$\ddot{a}^t_{\bar{s}} \geq \frac{\sum_{\bar{\pi}\in\bar{\Pi}} \ddot{a}^t_{\bar{\pi}\bar{s}}}{|\bar{\Pi}_{\bar{s}}|},\ \forall\ t \in \mathcal{T}, \bar{s} \in \bar{S} \tag{4.13}$$

$$\ddot{a}^t_{\bar{\pi}\bar{s}} \leq \sum_{i\in\mathcal{I}^t} \sum_{l\in L_i} \sum_{\bar{p}\in\bar{P}} w_{\bar{\pi}\bar{s}\bar{p}} \times \hat{y}^{ti}_{l\bar{p}}, \forall\ t \in \mathcal{T}, \bar{s} \in \bar{S}, \bar{\pi} \in \bar{\Pi}_{\bar{s}} \tag{4.14}$$

$$\ddot{a}^t_{\bar{\pi}\bar{s}} \geq \frac{\sum_{i\in\mathcal{I}^t} \sum_{l\in L_i} \sum_{\bar{p}\in\bar{P}} w_{\bar{\pi}\bar{s}\bar{p}} \times \hat{y}^{ti}_{l\bar{p}}}{\sum_{i\in\mathcal{I}^t} \sum_{l\in L_i} \sum_{\bar{p}\in\bar{P}} \hat{y}^{ti}_{l\bar{p}}}, \forall\ t \in \mathcal{T}, \bar{s} \in \bar{S}, \bar{\pi} \in \bar{\Pi}_{\bar{s}} \tag{4.15}$$

Constraints (4.1) and (4.2) ensure that all VNFs and virtual links for the active SFCs are embedded. Next, (4.3) and (4.4) ensure that VNFs are embedded on servers and NFV-PoDs that can support them and do not violate placement constraints for the VNF type and NFV-PoD. Then constraint (4.5) and (4.6) make sure that the endpoints of virtual link embedding on physical path and VNF embedding on physical server match with each other. Next, constraint (4.7) and (4.8) denote the capacity constraints for the servers and links, respectively. Constraint (4.9) represents the end-to-end latency constraint for the SFC. Constraint (4.10) ensures that a server is not in the active state when any VNF is not embedded on it, and constraint (4.11) ensures that a server is activated when there is at least one VNF embedded on it. Similarly, (4.12) and (4.13) make sure that a switch is in the active state when any active path passes through it, otherwise it will be in sleep mode. Finally, (4.14), and (4.15) ensure that switch ports are activated only when there is a virtual link embedded on a physical path that passes through that port.

4.4.6 Objective Function

Here, we calculate the total carbon footprint of the network. First, we define the function $d(t, t+1)$ to denote the time-duration between time-instances t and $t+1$. Now, for any time-duration $d(t, t+1)$ and a NFV-PoD $c \in \mathcal{C}$, the total energy consumption is calculated as follows:

SFC Migration:

An SFC is selected for migration when the new embedding reduces the cost by a certain proportion, specified by the migration threshold ϑ.

Energy Consumption of an SFC:

The energy consumption for active SFCs has three components: energy consumption by (i) physical servers, (ii) physical switches and ports, (iii) inter-NFV-PoD or backbone links. We measure the total energy consumption for the active SFCs as follows:

$$E_a^{tc} = \ddot{a}_i^t \times \sum_{i \in \mathcal{I}_n^t} \Bigg(\sum_{\bar{n} \in \bar{N}^c} \Big(e_{\bar{n}}^s \times (1 - \ddot{a}_{\bar{n}}^t) + \ddot{a}_{\bar{n}}^t \times \big(e_{\bar{n}}^b + (e_{\bar{n}}^m - e_{\bar{n}}^b) \times \sum_{n \in N_i} \sum_{r \in R} \hat{x}_{n\bar{n}}^{ti} \times \varphi_{f(n)}^r \times \xi^r \big) \Big)$$

$$+ \sum_{\bar{s} \in \bar{S}^c} \Big(e_{\bar{s}}^s \times (1 - \ddot{a}_{\bar{s}}^t) + \ddot{a}_{\bar{s}}^t \times \big(e_{\bar{s}}^b + \sum_{\bar{\pi} \in \bar{\Pi}_s} \ddot{a}_{\bar{\pi}\bar{s}}^t \times e_{m(\bar{\pi})} \big) \Big) + \sum_{l \in L_i} \sum_{\bar{p} \in \mathcal{B}(\bar{P}^c)} \hat{y}_{l\bar{p}}^{ti} \times \beta_l \times \xi^\beta \Bigg) \times d(t, t+1) \tag{4.16}$$

Now, the total brown energy consumption for time-duration $[t, t+1)$ and NFV-PoD c is equal to:

$$E^{tc} = \max(E_a^{tc} - e_g^{tc},\ 0)$$

Finally, the carbon footprint during a time-duration $[t, t+1)$ for the whole network is:

$$K^t = \sum_{c \in \mathcal{C}} E^{tc} \times \zeta_c$$

Our objective is to minimize the above equation over all time-instances. So, the objective function of the MLSO problem can be states as:

$$\textit{minimize} \ \sum_{t \in \mathcal{T}} K^t \tag{4.17}$$

Computing the optimal solution for large scale networks will require a substantial amount of time as the MLSO problem is NP-Hard. Hence, solving for the optimal solution is not a suitable approach in an online setting, where SFC requests must be embedded within seconds. In the following section, we present three heuristic algorithms that break down the MLSO problem into three subproblems and provide very fast near-optimal solutions.

4.5 Heuristics for Orchestration

In this section, we describe three heuristic algorithms for SFC embedding, consolidation, and migration. After receiving a new SFC request the heuristic for embedding is used to determine the mapping for the VNFs and inter-VNF links in the SFC. SFC migration and consolidation decisions are taken based on events like (i) changes in the availability of renewable energy and (ii) when one or more SFCs leave the system. We first describe the heuristic for SFC embedding. After that we present the heuristic algorithm used to take migration decisions. Finally, we present the heuristic for resource consolidation.

4.5.1 SFC Embedding

The SFC embedding algorithm is a three-stage heuristic. Stages one and three are executed by the G-Or, and stage two is executed by the L-Or(s) who are selected by the G-Or to host the service chain. The G-Or receives the SFC request as input and determines a set of NFV-PoDs as candidates to eventually host the service chain. An example chain is shown in Figure 4.8. The output of the embedding algorithm is a mapping for each VNF and inter-VNF links in the chain to a set of NFV-PoDs and physical links within and across these NFV-PoDs. The three stages of the embedding heuristic are as follows:

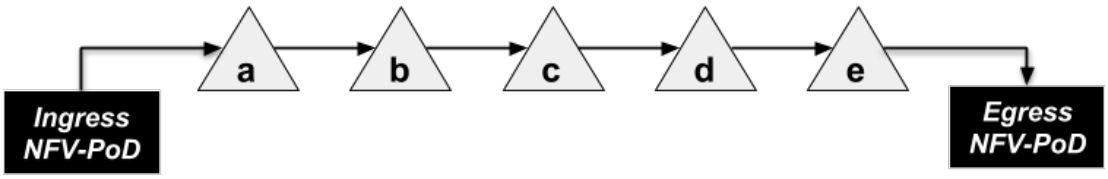


Figure 4.8: Example of a service chain

- **Stage-1:** This stage is executed by the G-Or. The G-Or calculates a set of potential paths between the ingress and egress NFV-PoDs of the SFC. Paths with a higher delay than the maximum allowed latency of the chain are immediately discarded. Among the remaining paths, the one with the maximum aggregate renewable energy is chosen. Then the G-Or sends the details of the SFC to each NFV-PoD (L-Or) on this path for initiating Stage-2.
- **Stage-2:** The L-Ors (chosen by the G-Or in the previous stage) run the second stage of the heuristic. Each L-Or computes a cost-matrix representing the embedding cost

Table 4.1: Cost-matrix computed by L-Or

	a	b	c	d	e
a	c(a)	c(a-b)	c(a-b-c)	c(a-b-c-d)	c(a-b-c-d-e)
b	-	c(b)	c(b-c)	c(b-c-d)	c(b-c-d-e)
c	-	-	c(c)	c(c-d)	c(c-d-e)
d	-	-	-	c(d)	c(d-e)
e	-	-	-	-	c(e)

Table 4.2: Embedding table for tabu search

	a	b	c	d	e
U	0	0	0	0	0
V	1	0	0	0	0
W	0	1	1	0	0
X	0	0	0	1	0
Y	0	0	0	0	1
Z	0	0	0	0	0

of sub-chains of the original SFC. An example cost-matrix for the SFC in Figure 4.8 is shown in Table 4.1. Each cell of the cost-matrix represents the cost of embedding a particular sub-chain of the original chain. The L-Ors return the computed cost-matrix to the G-Or for further processing.

- **Stage-3:** The final stage of the heuristic is executed by the G-Or. Upon receiving the cost-matrices from the NFV-PoDs selected in Stage-1, the G-Or constructs an embedding table as shown in Table 4.2. Each cell of this table represents the mapping of a VNF (on the column) to a NFV-PoD (on the row). Each cell will either contain a zero or one. In the final mapping, each column will contain exactly one one. A row can contain multiple ones, denoting the embedding of more than one VNF on the same NFV-PoD. There can be an exponential number of possible embeddings with respect to the number of rows; therefore, the final embedding is determined by using *Tabu Search* to selectively explore the search-space of all possible embeddings within a reasonable amount of time.

Algorithm 4, demonstrates the process of path selection in Stage-1. The algorithm takes as input the service chain s and the number of paths k to consider between the ingress and egress NFV-PoDs. The second parameter is used for finding the k-shortest

paths between the NFV-PoDs (Line 1). A small value of k reduces the running time of the algorithms, but restricts the search space of embedding paths. A large value of k increases the chance of finding a better embedding at the cost of increased processing time. During our performance evaluation, the value of k if varied between 3 to 7. Next, line 2 removes any path with a higher delay than the maximum tolerable delay of the chain. After that, the aggregate renewable energy of each path is computed, and finally, the path with the maximum renewable energy is returned.

Algorithm 4 Embedding: Stage 1 (runs @G-Or)

Require: service chain, s; number of considered paths between ingress and egress NFV-PoDs, k

Ensure: path p for embedding s

1: $P \leftarrow kShortestPaths(s.ingress, s.egress, k)$
2: Remove any path $p \in P$ `s.t.` $p.\delta > s.\delta$
3: **for all** `path` $p \in P$ **do**
4: **for all** `NFV-PoD` $c \in p$ **do**
5: $p.e_r \leftarrow p.e_r + c.e_r$
6: **end for**
7: **end for**
8: $sort(P)$ {In decreasing order of renewable energy}
9: **return** $P[1]$

The computation performed in Stage-2 is shown in Algorithm 5. It is a Dynamic Program (DP) to calculate the embedding cost of sub-chains (chain suffixes) in a bottom-up manner. Line 3 works as the base case for the DP. Each $cm[i][i]$ entry of the cost-matrix is initialized with the cost of embedding the $i-$th VNF in the chain. Next line 5 is the recursive call for computing embedding cost of a chain longer by one VNF in-terms of pre-computed costs. The embedding cost is computed based on a first-fit strategy, where the servers are pre-sorted according to their ids. Then, servers are considered in increasing order of id and the first server with enough capacity to host a VNF is chosen as the target server. The size of the cost-matrix is $n \times n$, where n is the length of the service chain. Algorithm 5 needs to fill the upper diagonal of the matrix, so there are $\frac{n(n+1)}{2}$ entries in total. For each embedding, the above-mentioned process might need to check all servers. Assuming that there are k servers within a single NFV-PoD, the running time of Algorithm 5 is $O(kn^2)$ that is polynomial in size of the input chain and intra-NFV-PoD network.

The third and final part of the heuristic is presented in Algorithm 6. In this stage, we

Algorithm 5 Embedding: Stage 2 (runs @L-Or)

Require: service chain, s;
Ensure: cost-matrix for s
1: initialize cm
2: **for** $i = 1$ **to** $s.size$ **do**
3: $cm[i][i] = embeddingCost(s[i])$
4: **for** $j = i + 1$ **to** $s.size$ **do**
5: $cm[i][j] \leftarrow cm[i][j-1] + embeddingCost(s[j])$
6: **end for**
7: **end for**
8: **return** cm

assume that chain embedding always progresses in the *forward direction* on the selected path. So, for the path `U->V->W->X->Y->Z`, if the second VNF `b` is embedded on NFV-PoD `W`, then the subsequent VNFs (*i.e.*, `c, d, e`) cannot be embedded on any NFV-PoD that comes before `W` on the path (*i.e.*, `U` and `V`); they must be embedded on either `W` or NFV-PoDs that come after `W`, *i.e.*, `X, Y,` and `Z`. Algorithm 6 first initializes an embedding-table (*e.g.*, Table 4.2) by filling all the cells with zeros. Then, line 3, fills the embedding table with suboptimal solution based on the first-fit approach. Here, each VNF is embedded on the first NFV-PoD that has enough capacity maintaining the ordering constraint mentioned above. After that, we perform a tabu search on the embedding-table to find a better solution. The steps of the tabu search are described below:

Initial Solution:

In line 3, the initial solution is generated based on the first-fit approach; the algorithm attempts to embed the next VNF in the next NFV-PoD on the path. It maintains the constraint that an SFC's links always move in the forward direction on the path. If the first-fit approach fails to generate a valid solution, then the initial solution is generated by randomly embedding VNFs on any NFV-PoD with enough capacity (line 5).

Solution Neighborhood:

Each iteration of the tabu search moves from the current solution to a neighboring solution. Here, solutions are defined by the values in the embedding-table, hence, one solution is the neighbor of another solution if they differ in exactly one column, *i.e.*, the embedding

Algorithm 6 Embedding: Stage 3 (runs @G-Or)

Require: cost-matrix for s from each NFV-PoD on path p;
Ensure: mapping of VNFs and inter-VNF links in s to NFV-PoDs on path p

$S \leftarrow \phi$ {S: current solution}
if $firstFitSolution()$ `is valid` **then**
  $S \leftarrow firstFitSolution()$
else
  $S \leftarrow randomSolution()$
end if
$S^* \leftarrow S$ {S^*: best solution so far}
while `stopping criteria not met` **do**
  $SS \leftarrow \phi$
  for $S^c \in N(S)$ **do**
    if $S^c \notin T$ **then**
      $SS \leftarrow SS \cup S^c$
    end if
  end for
  $S \leftarrow best(SS)$
  $add(T, S, S^*)$
  if $cost(S) < cost(S^*)$ **then**
    $S^* \leftarrow S$
  end if
  $update(T)$
end while
return S^*

location of one VNF. The key element for the performance of tabu search is the fast computation of costs for a solution. Here, we can calculate the cost of an embedding quickly from the cost-matrices calculated in the previous step. The computation of cost reduces to at most m lookups from cost-matrices where m is the number of VNFs in the service chain.

Tabu List:

A list of solutions or rules which are deemed tabu (forbidden) for a specific amount of time, enables the tabu search meta-heuristic to avoid getting stuck at locally optimal solutions. Here, if a VNF v is moved away from a server s, then the pair $\langle v, s \rangle$ is added to the tabu list T (line 16 or Algorithm 6). The item stays in the list for $m - 1$ iterations, which gives other VNFs the opportunity to change their embedding locations. The amount of time each rule stays in the tabu list is updated at line 20.

Stopping Criteria:

If the best solution (S^*) does not improve for $m + n$ iterations, where m and n are the number of VNFs and NFV-PoDs, then the tabu search is terminated.

4.5.2 SFC Migration

SFC migration decisions are also made using the tabu search algorithm described above. The algorithm first builds a list of potential paths between the ingress and egress NFV-PoDs and then orders them according to the number of overlapping NFV-PoDs with the current path. These paths are monitored for changes in renewable energy and available resource. Whenever there is an increase in embedding cost or availability of new resource at one or multiple NFV-PoDs on these paths, the embedding-table with updated costs is used to perform another tabu search for the candidate path. This approach restricts migrations on a subset of the possible paths but provides a fast and simple algorithm for taking migration decisions. An SFC migration decision is made based on the value of the parameter migration threshold, ϑ, which is an input to the algorithm. ϑ specifies the percentage gain in-terms carbon footprint that must be achieved when migrating an SFC to a new path. More specifically, if ϑ is set to `0.2`, then an SFC is migrated to a new path only when it reduces the embedding cost by at least 20%. This approach provides network operators a tunable parameter to control the migration of SFCs.

4.5.3 SFC Consolidation

The consolidation heuristic is run locally by the L-Or at each NFV-PoD, after an SFC's lifetime expires and resources allocated for that SFC are freed. Consolidation is performed by moving VNFs towards one side (represented by sequential numbering of the servers) of the server racks of the NFV-PoD. We assume that each server in a NFV-PoD is numbered and the numbering starts from one side and increases towards the other side of the server racks. The consolidation algorithm moves VNFs towards lower numbered servers and updates their routing paths accordingly. This problem is similar to the bin-packing problem, where VNFs on a candidate server to be turned off are considered to be the items and the rest of the active servers are considered as bins with different capacities. Here, we use a modified version of the Best Fit Decreasing (BFD) [151] algorithm to find a solution. The BFD algorithms is known to perform reasonably well for bin packing problems[152]. The algorithm is shown in Algorithm 7.

The consolidation algorithm takes the current resource allocation state as input and determines a sequence of VNF moves for resource consolidation. If all VNFs from a server can be moved to other active servers, then that server is put in sleep mode. In line 1 and 2, the set S is initialized with all active servers, then the set is sorted in decreasing order of server ids, as we want to move VNFs from servers with higher ids to servers with lower ones. In the for loop starting at line 3, each server is considered in decreasing order of id, and in the for loop between lines 6 and 21, all VNFs currently hosted on a server are checked and the destination server that minimizes energy consumption (line 9 to 14) is selected. This information is saved in the $vnfDests$ data structure. $vnfDests$ is implemented as a `map` that saves the mapping from a VNF to its new destination server's numerical id. If all VNFs hosted on a server can be moved to some other servers with lower ids, then $vnfDests$ is used to move out all VNFs from server s, and then server s is put into sleep mode (line 23 and 24) to save energy.

4.6 Performance Evaluation

We perform trace-driven simulations on different network topologies and SFC request patterns to compare and contrast the performances of our proposed algorithms. In the following, we first describe the datasets in Section 4.6.1, then the performance metrics in Section 4.6.2, followed by the simulation setup in Section 4.6.2. Finally, a detailed discussion of the simulation results is provided in Section 4.6.5.

Algorithm 7 Resource consolidation

Require: resource allocation state of the NFV-PoD

1: $S \leftarrow$ active servers
2: $sortDecreasingID(S)$
3: **for all** `server` $s \in S$ **do**
4: $canMoveAllVNFs \leftarrow$ **true**
5: $vnfDests \leftarrow \phi$
6: **for all** `VNF` v `hosted on` s **do**
7: $minPower \leftarrow MAX$
8: $selectedServer \leftarrow NULL$
9: **for all** `server` $t \in S$ **and** $t.id < s.id$ **do**
10: **if** $t.residual >= v.req$ **and** $t.power(v) < minPower$ **then**
11: $minPower \leftarrow t.power(v)$
12: $selectedServer \leftarrow t$
13: **end if**
14: **end for**
15: **if** $selectedServer = NULL$ **then**
16: $canMoveAllVNFs \leftarrow$ **false**
17: break
18: **else**
19: $vnfDests \leftarrow vnfDests \cup \{v, selectedServer\}$
20: **end if**
21: **end for**
22: **if** $canMoveAllVNFs =$ **true then**
23: $moveVNFs(s, vnfDests)$
24: put s in sleep mode
25: **end if**
26: **end for**

4.6.1 Datasets

SFC Data

We generate SFCs of length between three and six consisting of a pre-determined set of VNFs from Table 4.3. Due to the lack of any published real-world data on the type and length of SFCs, we resort to a synthetic generation method. We surveyed the relevant research literature [37, 153, 43, 154, 155] and Internet drafts [156] to determine the set of VNFs, and the length of SFCs to use in our simulations. VNF characteristics listed in Table 4.3 were obtained by studying the relevant literature and product data sheets [113, 45, 153].

SFCs are generated between random source-destination locations according to a Poisson process with an arrival rate of λ. The duration of each SFC is determined based on an exponential distribution with a mean of $1/\mu$. The specific values of λ and μ are determined based on the objectives of a particular simulation. The bandwidth demands of the SFCs are generated according to a sin cyclostationary traffic matrix sequence generated according to the process described in [157]. The mean and standard deviation of traffic volume is set to 500mbps and 0.9, respectively. The maximum allowable delay for an SFC is uniformly distributed between 100 milliseconds to one second.

Power Consumption Data

The power consumption data for the server, switch, and VNFs are listed in Table 4.3.

Renewable Energy Data

The publicly available renewable energy data from the U.S. Climate Reference Network and Regional Climate Reference Network [60] for June 2017 is used in this work. The data provide an estimate of energy generation potential from solar and wind for a large number of U.S. cities in different states. NFV-PoDs are assumed to host solar panels sizes varied randomly between four to eight square meters. For wind energy, the number of wind turbines is varied randomly between one to three.

Network Topology

We use Point-of-Presence (PoP) level topologies of four ISPs obtained from the RocketFuel topology dataset [109]. The node and edge counts, and the maximum latency of any edge

Table 4.3: Server, switch, and VNF power consumption

Server Data [102]		
Physical CPU Cores	Idle Energy	Peak Energy
16	80.5W	2735W
Switch Data [143, 59]		
Dell PowerConnect	Sleep Mode Energy	Active Mode Energy
	10W	110W
Port Link Rate	1 Gbps	10 Gbps
	1.2W	4.3W
VNF Data [113, 45]		
Network Function	CPU Required	Processing Delay
Firewall 1	4	0.5 ms
Firewall 1	2	0.8 ms
Proxy 1	4	0.025 ms
Proxy 2	8	0.001 ms
NAT 1	8	0.1 ms
NAT 2	16	0.05 ms
IDS	4	0 ms

Table 4.4: Network topologies

Topology	Node Count	Edge Count	Maximum Edge Latency
AS-13129	7	9	3 ms
AS-7170	18	60	43 ms
AS-3549	61	486	66 ms
AS-3561	92	329	60 ms

for these topologies are reported in Table 4.4. The PoP locations in these topologies are assumed to represent potential locations for deploying NFV-PoDs. Inside each PoP, we assumed the presence of a fat-tree like data center network topology consisting of switches, gateway router, and servers. For the sake of simplicity, we assume a linear power consumption model for the servers *i.e.*,the amount of consumed power increases linearly with the number of CPUs [158].

4.6.2 Performance Metrics

Carbon Footprint

Carbon footprint is measured for all nodes and edges in the network. We report the carbon footprint by summing the carbon footprint for each NFV-PoD and the inter-NFV-PoD edges. So, even if an NFV-PoD is not participating in embedding a particular SFC, its contribution is summed to calculate the overall carbon footprint (Eq. 4.17). This approach ensures a fair comparison between the optimal and heuristic solutions.

Renewable Energy Usage

This metric measures how much renewable energy is utilized at each time-instance by the optimal and heuristic solutions. Even though the utilization of renewable energy is not a direct objective of the addressed problem, this metric is optimized indirectly, as more renewable energy leads to less carbon footprint. We report the proportion of renewable and brown energy usage to measure how effectively the proposed algorithms utilize available energy sources.

SFC Acceptance Ratio

We measure SFC acceptance ratio as the ratio of embedded SFCs to the total number of received SFC requests. Acceptance ratio is correlated with the overall carbon footprint; a lower acceptance ratio will obviously reduce the overall carbon footprint. Therefore, during the simulations, datasets are pre-processed to avoid the impact of one of these metrics on the other one. When measuring carbon footprint, datasets are per-processed to ensure that both optimal and heuristic solutions can embed 100% of the SFC requests; thereby, nullifying the impact of acceptance ratio. In turn, when comparing acceptance ratios for optimal and heuristic solutions, it is ensured that an SFC request is not rejected when there is adequate free resources in the network. SFC requests are embedded in a sequential manner while the optimizer (optimal or heuristic) minimizes the overall carbon footprint for each SFC.

4.6.3 Migration and Reduction in Carbon Footprint

The impact of SFC migration across NFV-PoDs is measured by calculating the percentage reduction in carbon footprint. Two identical simulations are executed with and without enabling migration and then the reduction within each hour is measured to demonstrate the impact of migration on carbon footprint.

4.6.4 Simulation Setup

We perform extensive simulations to show the effectiveness of the proposed algorithms in terms of overall carbon footprint, utilization of available renewable energy, and acceptance ratio. In addition, the impact of migration on the overall carbon footprint is also reported. In all experiments, we simulate 24 hours of inbound and outbound SFC requests. Renewable energy at the NFV-PoDs is simulated by assigning them to different US cities obtained from the renewable energy data [60]. The mathematical formulation of the MLSO problem is implemented as an ILP using the IBM CPLEX Optimizer v12.5 [159]. All simulations are conducted on a machine with a six core AMD FX-6300 1.4 GHz processor and 16 GB of memory running Ubuntu 14.04. The heuristic is implemented in C++, and a discrete-event simulator is developed in Python to invoke either CPLEX or heuristic implementations to run the simulations.

4.6.5 Results

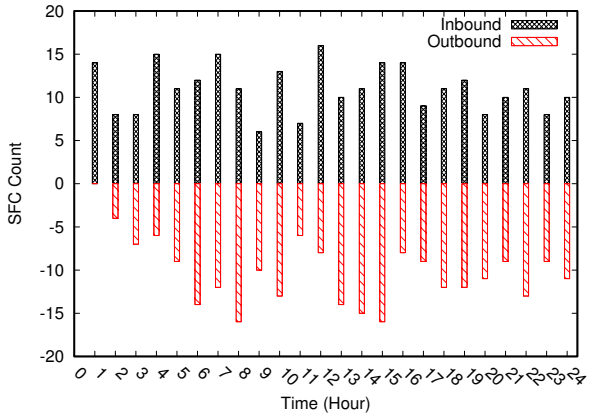
Overall Carbon Footprint

In these simulations, the arrival rate of new SFC requests (λ) is set to 10 requests per hour, and the average lifetime of an SFC (μ) is set to 1.5 hours. The traffic patterns and overall carbon footprint for AS-13129 is shown in Figure 4.9. The number of new (inbound) and expired (outbound) SFC requests are shown in Figure 4.9(a). Here, a positive number represents inbound SFCs, and a negative number represents outbound SFCs. As we can see from the figure, the average number of inbound SFC requests is around 10. The expiry event of an SFC depends on its lifetime; therefore, the number of outbound SFCs does not show any pattern and is mostly arbitrary. The number of active SFCs is shown in Figure 4.9(b). An SFC is considered to be in the active state between its arrival and departure time-instances. New SFC requests at a time-instance are considered for embedding, while already existing (that arrived at a previous time-instance) SFCs are considered for migration.

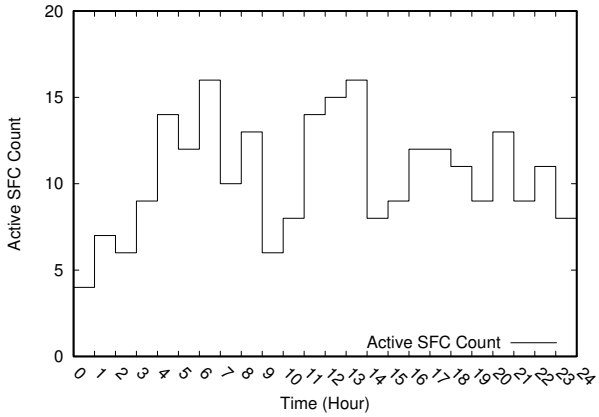
Figure 4.9(c) shows the overall carbon footprint for the optimal and heuristic solutions. The carbon footprint obtained from the heuristic solution is close to that of the optimal solution for all time-instances. The maximum deviation is observed at time-instance 7, where the heuristic solution's cost is 1.3 times the cost of the optimal solution. The increase and decrease in the overall carbon footprint for both heuristic and optimal solutions show a similar pattern, which demonstrates the adaptability of the heuristic algorithms with traffic fluctuations. The heuristics for embedding, consolidation, and migration work together to achieve near-optimal carbon footprint for the network.

The results for AS-7170 is shown in Figure 4.10. The inbound and outbound SFC counts are shown in Figure 4.10(a), and the number of active SFCs over the 24 hours is shown in Figure 4.10(b). The overall carbon footprint for the entire network is shown in Figure 4.10(c). AS-7170 has a higher proportion of edges compared to AS-13129. A higher number of edges increases the number of possible paths to embed an SFC. The optimal solution always chooses the least-cost path by paying the price of higher running time. However, as presented in the following section, the heuristic algorithm provides near-optimal solutions without any noticeable sacrifice in running time. The performance gap is very similar to that obtained for the previous experiment.

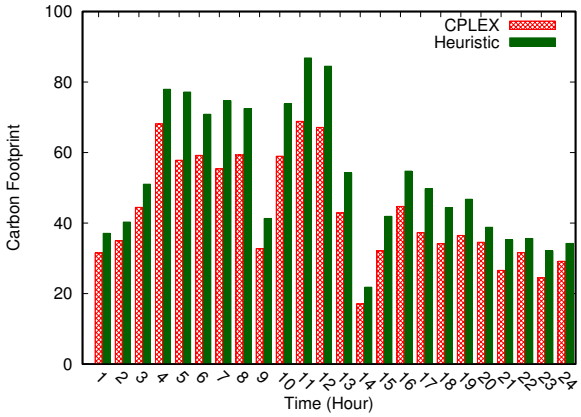
For AS-3549 and AS-3561, the CPLEX implementation could not find the optimal embedding for a single SFC even after an hour. Hence, we refrained from running these experiment for the optimal solution. Figure 4.11 and Figure 4.12 report the simulation results of the heuristic algorithm for AS-3459 and AS-3461, respectively.



(a) In- and out-bound traffic (10 reqs/hour)

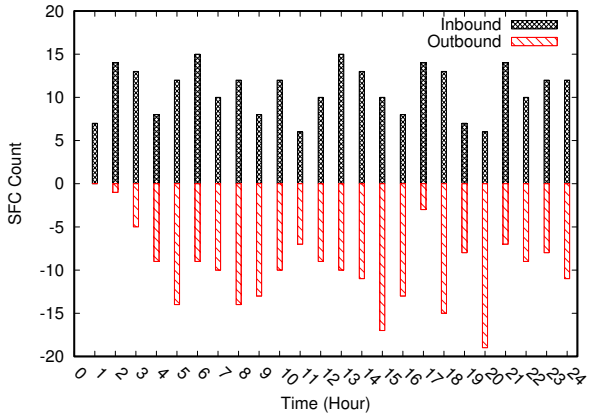


(b) Active SFC counts

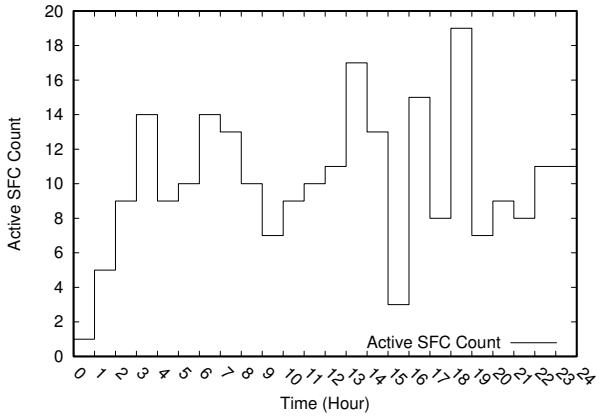


(c) Overall carbon footprint

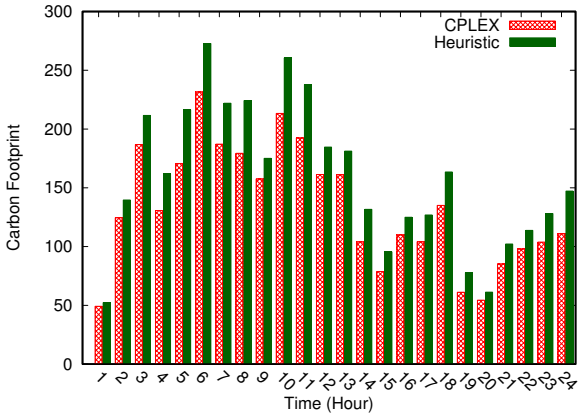
Figure 4.9: Traffic pattern and overall carbon footprint for AS-13129



(a) In- and out-bound traffic (10 reqs/hour)

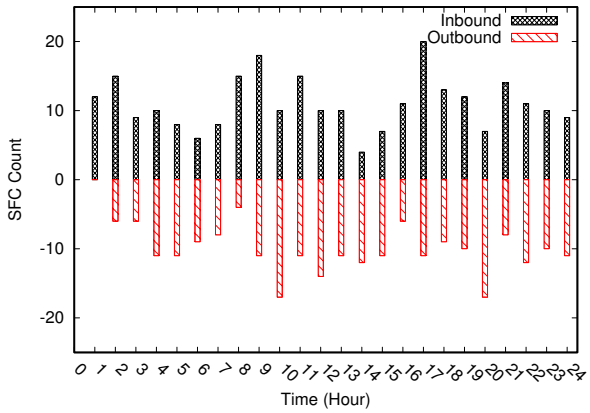


(b) Active SFC counts

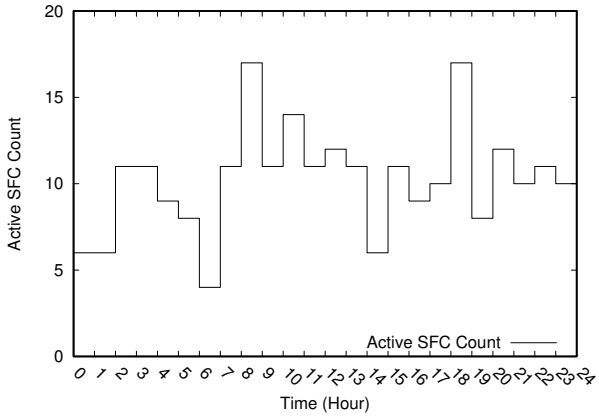


(c) Overall carbon footprint

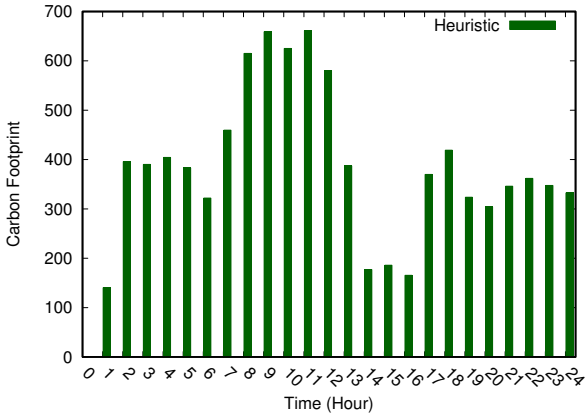
Figure 4.10: Traffic pattern and overall carbon footprint for AS-7170



(a) In- and out-bound traffic (10 reqs/hour)

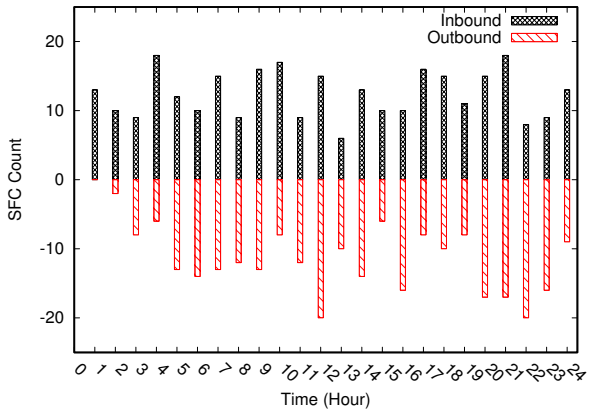


(b) Active SFC counts

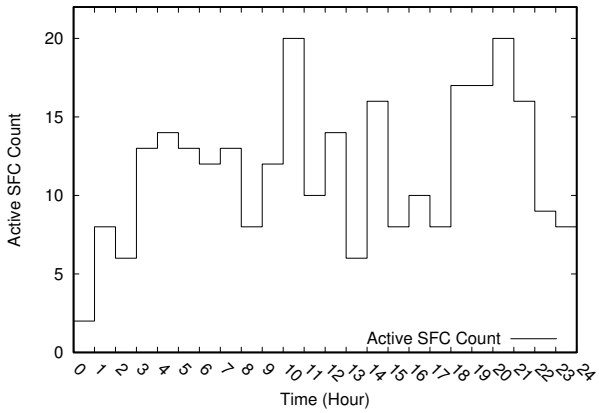


(c) Overall carbon footprint

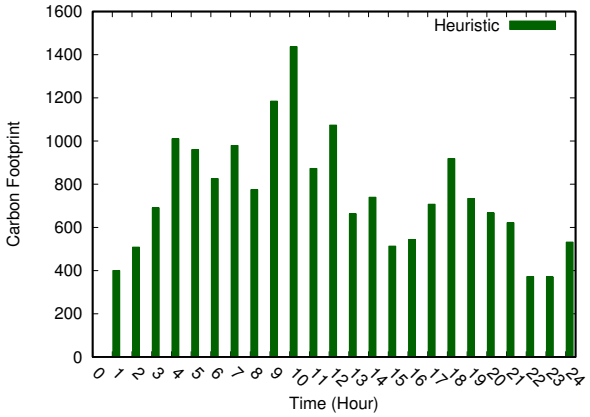
Figure 4.11: Traffic pattern and overall carbon footprint for AS-3549



(a) In- and out-bound traffic (10 reqs/hour)



(b) Active SFC counts



(c) Overall carbon footprint

Figure 4.12: Traffic pattern and overall carbon footprint for AS-3561

Table 4.5: Running time of heuristic (per SFC request)

Topology	Min	5^{th} %tile	Mean	Median	Mode	95^{th} %tile	Max
AS-13129	2.0 ms	2.0 ms	4.4 ms	4.0 ms	3.0 ms	7.0 ms	8.0 ms
AS-7170	3.0 ms	3.0 ms	4.5 ms	5.0 ms	3.0 ms	7.0 ms	8.0 ms
AS-3549	4.0 ms	4.0 ms	5.5 ms	5.0 ms	5.0 ms	9.0 ms	12.0 ms
AS-3561	4.0 ms	4.0 ms	5.9 ms	5.0 ms	5.0 ms	13.0 ms	17.0 ms

Table 4.6: Running time of CPLEX (per SFC request)

Topology	Min	5^{th} %tile	Mean	Median	Mode	95^{th} %tile	Max
AS-13129	1.2 s	2.1 s	4.4 s	3.6 s	2.1 s	7.5 s	13.2 s
AS-7170	8.0 s	27.5 s	1 m 28.3 s	1 m 15.5 s	1 m 2.5 s	2 m 55 s	4 m 16.5 s
AS-3549	> 1 h	-	-	-	-	-	-
AS-3561	> 1 h	-	-	-	-	-	-

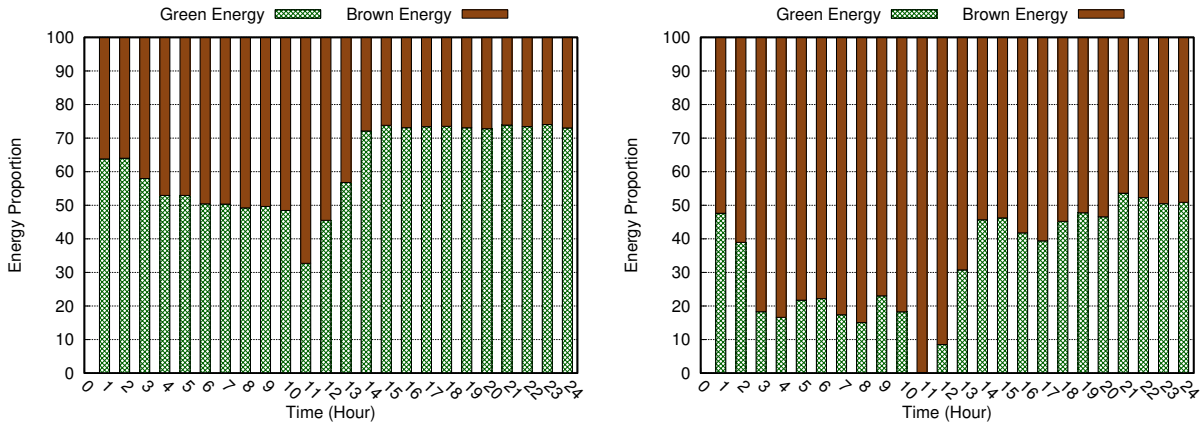
Running Time of Optimal and Heuristic Algorithms

Running time statistics of the optimal and heuristic algorithms for a single SFC is reported in Table 4.5 and Table 4.6, respectively. AS-13129 is the smallest topology included in the simulations; it has 7 nodes and 9 edges. The optimal solution requires a minimum of 1.2 seconds to embed an SFC, while the mean and maximum time required are 4.4 and 13.2 seconds, respectively. Other statistics like mode, median, 5^{th} and 95^{th} percentiles are also reported in Table 4.6. In contrast, the heuristic requires a maximum of only 8 milliseconds to embed an SFC, which is a 150 times improvement compared to even the minimum time required for the optimal solution. Additional relevant statistical data are reported in Table 4.5. Now, for AS-7170 (18 nodes and 60 edges) the optimal solution requires 1 minute and 28 seconds on average, whereas the heuristic requires only 4.5 milliseconds, an improvement of around 20000 times. The maximum time required by the heuristic is 8 milliseconds, whereas the minimum time required by the optimal solution is 8 seconds.

We ran the CPLEX code on AS-3549 and AS-3561; however, it could not even generate the first feasible solution. However, the heuristic took only 5.5 and 5.9 milliseconds on average for AS-3549 and AS-3561, respectively.

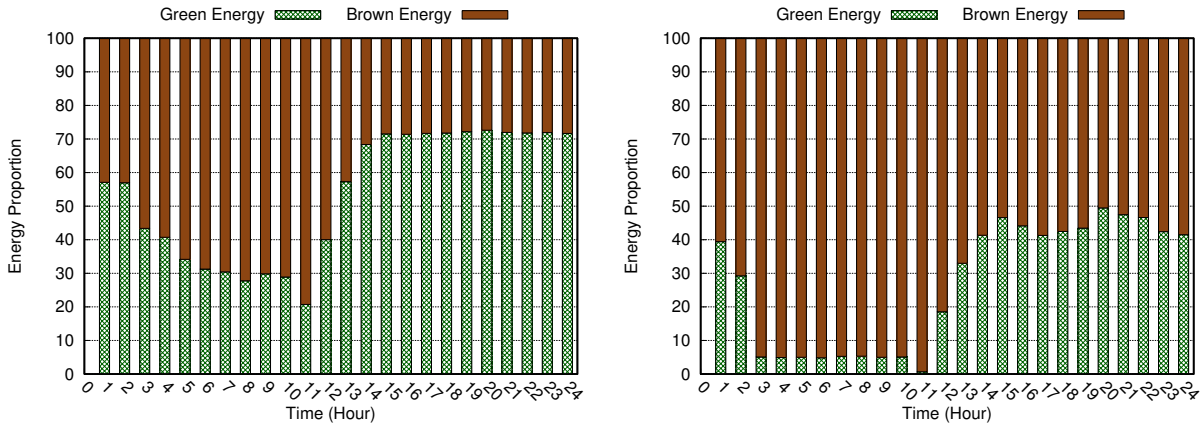
Utilization of Renewable Energy

Renewable energy utilization is calculated as the proportion of renewable energy that is utilized within an hour for running all servers and switches in the network. For each hour,



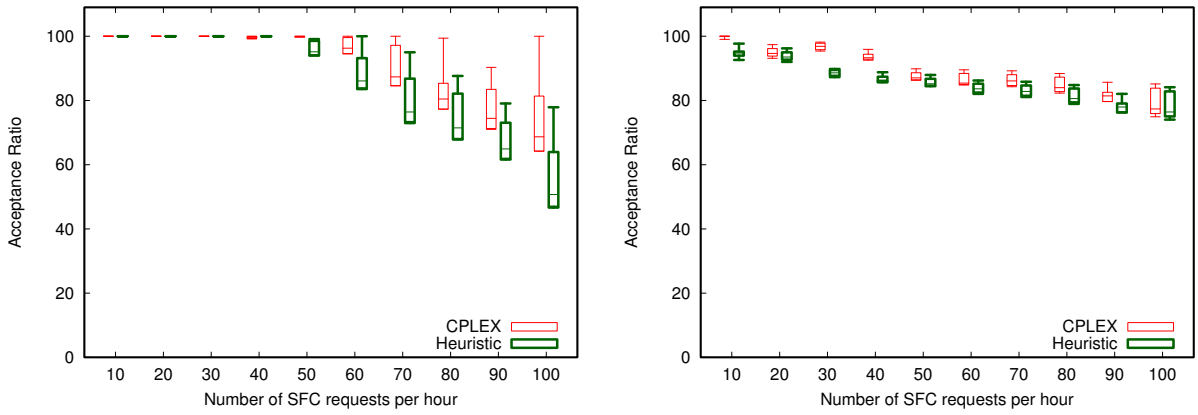
(a) Optimal solution (b) Heuristic solution

Figure 4.13: Green energy utilization (AS-13129)



(a) Optimal solution (b) Heuristic solution

Figure 4.14: Green energy utilization (AS-7170)



(a) AS-13129 (b) AS-7170

Figure 4.15: Acceptance ratio

we measure the total amount of renewable and brown energy used by the network, then the proportion of renewable energy is calculated as the ratio of renewable energy to the total (renewable + brown) energy.

Figure 4.13(a) and Figure 4.13(b) show the renewable and brown energy proportions for AS-13129 for optimal and heuristic solutions, respectively. The optimal solution achieves much better renewable energy utilization than the heuristic. The primary cause behind this issue is the way the heuristic chooses a path to embed an SFC. The heuristic chooses a path that offers the highest amount of renewable energy; however, the total renewable energy might be concentrated only on one or a few NFV-PoDs on the path. Choosing such a path can reduce the carbon footprint of the servers, but significantly increases the carbon footprint for embedding the SFC's edges. This behavior is discovered by closely examining the embedding locations of the optimal and heuristic solutions. The performance of the heuristic can be improved by examining more paths; however, this approach increases the running time and does not improve performance in all cases. Therefore, we opted for a lower running-time and simpler heuristic algorithm by choosing the current approach of path selection. The renewable and brown energy proportions for the optimal and heuristic in AS-7170 are shown in Figure 4.14. The heuristic shows similar performance for this topology as well.

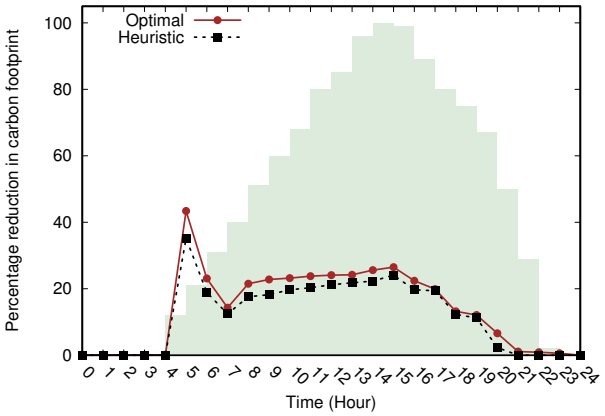


Figure 4.16: Impact of migration (AS-13129)

Acceptance Ratio

In these simulations, we increase the number of inbound SFC requests from 10 to 100 requests per hour. At each step, we simulate 24 hours and track the percentage of SFCs that are successfully embedded within each hour. The acceptance ratio is calculated per hour, and then the min, 5^{th}-percentile, mean, 95^{th}-percentile, and the max values are plotted for the optimal and heuristic solutions in Figure 4.15(a) and Figure 4.15(b) for AS-13129 and AS-7107, respectively. We can see from the figure that the acceptance ratios for the heuristic solution are very close to those of the optimal solution in both cases. For AS-13129, the acceptance ratio of the heuristic is usually within 12% of the optimal; except for the 100 request per hour setting. In case of AS-7170, the heuristic performs much better as there are more paths compared to AS-13129, which offer more embedding opportunities to the heuristic. For AS-7170, the acceptance ratio of the heuristic is within 7% of the optimal.

Impact of Migration

To measure the impact of migration, we run simulations with and without enabling migration of SFCs. We also pre-process the renewable energy data to start SFC embedding at a time-point when there is no renewable energy, and then the amount of renewable energy rises, and finally, it decreases. This approach represents a renewable energy source that shows the time-of-day effect, like a solar panel, and captures the impact of migration in three possible scenarios. The results for AS-13129 is shown in Figure 4.16. At each hour,

the normalized value of the overall renewable energy, and the percentage reduction in carbon footprint due to migration is plotted for both optimal and heuristic solutions. The overall renewable energy is calculated by summing the available renewable energy amounts at all NFV-PoDs.

The differences between the percentage reduction in carbon footprint for the optimal and heuristic solutions are very small. The initial rise in carbon footprint reduction is due to the fact that at hour 4 renewable energy becomes available for the first time, and a significant number of SFCs are migrated to reduce carbon footprint. However, the effect subsides after hour 5 as the only candidate SFCs that can reduce overall carbon footprint are the ones that arrived during hour 4. After that, the percentage reduction increases with increasing renewable energy. However, when the amount of renewable energy starts decreasing, the percentage reduction in carbon footprint also keeps decreasing. At this stage, the only opportunity to reduce carbon footprint occurs when some SFCs expire, and resources powered by renewable energy become available. The performance of optimal and heuristic solutions is close. Similar results are obtained for AS-7170 and are not included for the sake of brevity.

4.7 Related Work

SFC orchestration has received much attention in recent years [149, 43, 160, 161, 162, 163]. However, none of the previous works in the literature targeted the problem addressed in this work. A survey of SFC placement algorithms is provided in [149], where the authors have classified the existing algorithms under categories like basic, dynamic, online, multiple-provider, schedule, mobile network, and data center. However, none of the surveyed papers consider a problem similar to MLSO. The existing online algorithms consider placement of SFC requests but do not examine the migration of existing SFCs or consolidation of freed resources to reduce cost. Eramo *et al.* addressed a similar problem where SFCs are provisioned and migrated with the objective to reduce energy cost; however, their work does not consider renewable energy sources [160]. Besides, they assume that each server will pre-active a VNF of each type and their proposed algorithm determines the number of cores for each such pre-activated instance. However, this is not a suitable assumption as most VNFs have a fixed set of cores that are required for its operation.

The works in [43, 161, 163], focus on the general SFC orchestration problem to reduce energy consumption but do not consider the possibility of migration and utilization of renewable energy sources. In [160], the authors address the issue of migration; however, they do not consider the availability of renewable energy, and they assume that all SFC

requests are known in advance for their proposed heuristics. In essence, their algorithms are offline, where we consider the online version of the problem and propose heuristics for embedding and migration of SFCs, along with consolidation of resources. In [162], the authors propose a polynomial time heuristic for SFC embedding; however, they do not model the energy consumption of servers and switches, or migration of VNF chains. Their model tries to maximize the service provider's revenue by accepting more SFC requests without providing any details on how the provider's operational cost is calculated.

Authors in [118] proposed a grammar for specifying VNF chains and then provided a quadratic formulation for VNF chain placement. In contrast, we provide a linear formulation for SFC embedding across multiple locations and consider VNF migration between these locations to reduce carbon footprint. In [119], the authors provided an LP-relaxation based approach for finding inter-data center VNF chain placement. However, due to LP-relaxation, their solution violates physical resource capacities by a factor of 16. Our solutions do not have such issues, and we provide extensive simulations to show that our proposed heuristic achieves near-optimal performance within milliseconds. A genetic algorithm for VNF chain placement is proposed in [120], but it does not address the issue across multiple locations. An orchestration architecture for automatic VNF placement is proposed in [121], but the authors do not provide any particular algorithms for orchestration.

Green energy aware placement strategies are also explored in areas like Virtual Network (VN) and Virtual Data Center (VDC) embedding [104, 96]. The problem of VN embedding across multiple data centers to reduce the total energy consumption of switches and servers is considered in [164]. This work assumes that the workload of the VNs can be predicted in the future and takes future migration decisions based on the predicted values. It considers two time periods, workload during day and night while taking VN embedding decision. Moreover, this work does not consider the availability of renewable energy sources. In contrast, our algorithm is online and takes migration decisions based on the fluctuation of renewable energy in real-time.

Authors in [165] introduced FORTE, a framework that enables large-scale Internet application providers to navigate the three-way tradeoff between carbon footprint, electricity cost, and access latency. FORTE also provides green data center upgrade and expansion plans that can reduce carbon footprint. Authors in [166, 167, 168] propose VDC embedding algorithms across geographically distributed data centers while minimizing carbon footprint by utilizing green energy sources; however, these works do not consider migration of virtual nodes (or VNFs) to utilize green energy. They consider one-shot embedding of VDC requests across distributed data centers. Furthermore, they do not model the internal server and bandwidth of each data center separately. They consider data centers with infinite capacity and only consider the bandwidth of the backbone network while taking

embedding decisions. In contrast, we consider both inter- and intra-NFV-PoD networks and the server capacities inside each NFV-PoD while taking embedding, migration, and consolidation decisions.

4.8 Conclusion

In recent years telecommunication networks have experienced a monumental rate of expansion, and this rate is going to accelerate at an even higher pace in the future. A negative side-effect of this rapid expansion is the equally rapid rise of carbon footprint that is generated from the power consumption of telecommunication infrastructures. Government regulations and environmental concerns are pushing network operators to devise innovative solutions to minimize carbon footprint. The challenge for network operators is to achieve this without significantly increasing their operational cost. In this perspective, we propose ESSO, an SFC orchestrator that reduces the overall carbon footprint of a telecommunication network by intelligently embedding and migrating SFCs across different locations. In addition, ESSO consolidates recently freed resources to reduce energy consumption. The fundamental idea of ESSO is to generate a near-optimal SFC embedding across the infrastructure by piecing together partial embedding data collected from a small number of NFV-PoDs. We performed extensive simulations on four ISP topologies collected from the Rocketfuel dataset. Our results show that the proposed heuristic algorithm provides near-optimal solutions within milliseconds for all topologies.

Chapter 5

Conclusion and Future Research Directions

5.1 Conclusion

This thesis focused on resource allocation and orchestration techniques that are essential to realize the vision of network softwarization. The primary objective of softwarization is to overcome the current ossified state of network architecture, and evolve towards a more open, agile, flexible, and programmable networking paradigm that will reduce both CAPEX and OPEX, cut-down time-to-market of new services, and create new revenue streams. The fundamental idea of network softwarization is to achieve virtualization and programmability of network resources through the decoupling of processing logic that realizes network services from the underlying physical hardware that transports bits flowing through the network. Softwarization offers programmability, flexibility, and enhanced control over the network. To fully exploit the benefits of network softwarization, innovative techniques and mechanisms are required for all aspects of network management and control. The resource allocation and orchestration techniques presented in this thesis utilize the functionality provided by softwarization to reduce operational cost, improve resource utilization, ensure scalability, dynamically scale resource pools according to demand, and optimize energy usage from multiple sources. This thesis tackled three key resource allocation and orchestration challenges in network softwarization. These challenges and the contributions of the thesis are summarized below.

The first contribution of this thesis addressed the scalability issue of the centralized control plane in SDN. The SDN controller is the brain of the network and controls each

and every traffic flowing through the network. In most cases, a single controller is not enough to handle the processing, delay, and fault-tolerance requirements of a network. In these cases, multiple controllers must be deployed to ensure the necessary control plane performance requirements. One such requirement is the switch flow-setup time, and in this work, we identified and formulated the Dynamic Controller Provisioning Problem (DCPP) that minimizes flow-setup time by dynamically adapting the number of controllers and their locations according to demand fluctuations in the network. We proposed a management system for dynamically deploying multiple controllers. We also provided a mathematical formulation of DCPP as an ILP. Since DCPP is an $\mathcal{NP}$-hard problem, we provided two heuristic algorithms (DCP-GK and DCP-SA) to solve it. Our simulation results showed that our solution can achieve lower flow-setup times and minimal communication overhead compared to that of the static version of the problem (using single or multiple controllers). Our system achieved a balance between flow-setup time and messaging overhead. Evaluation results also showed that DCP-SA and DCP-GK succeeded in finding the right trade-off between these two extremes and provide near-optimal solutions. DCP-SA provides better results than DCP-GK but takes longer to converge.

The second contribution of this thesis focused on the issue of operational cost management of virtualized network services in the area of NFV. Virtualized network functions or VNFs provide a flexible way to deploy, operate and orchestrate network services with much less capital and operational expenditures. Recent technological advancements in software-based network packet processing have enabled VNFs to provide performance that is similar to that of a hardware appliances, *e.g.*, ClickOS, and OpenNF. Network operators are already opting for NFV based solutions. In this contribution, we developed a model for dynamic VNF orchestration that can be used to determine the optimal number and location of VNFs to minimize network operational cost and reduce resource fragmentation. We named this problem as the Virtualized Network Function Orchestration Problem or VNF-OP. We provided a mathematical formulation of the problem and proved the problem to be $\mathcal{NP}$-hard. We also devised a heuristic that provides near-optimal solution within a few seconds. Our trace-driven simulations on the Internet2 research network demonstrated that network OPEX can be reduced by a factor of 4 over hardware middleboxes through proper VNF orchestration. Specifically, we devised two solutions to the VNF orchestration problem: CPLEX based optimal solution for small networks and a heuristic for larger ones. We found that the heuristic produces solutions that are within 1.3 times of the optimal solution, yet the execution-time is about 65 to 3500 times faster than that of the optimal solution.

The third and final contribution of the thesis presented ESSO: An Energy Smart SFC Orchestrator that orchestrates service function chains across multiple deployment loca-

tions (NFV-PoDs) with the objective to reduce the carbon footprint of the network while considering both brown and renewable energy sources. In recent years telecommunication networks have experienced a monumental rate of expansion, and this rate is going to accelerate at an even higher pace in the future. A negative side-effect of this expansion is the rise of carbon footprint that is generated from the power consumption of telecommunication infrastructures. Government regulations and environmental concerns are pushing network operators to devise innovative solutions to minimize carbon footprint without significantly increasing their operational cost. In this respect, we introduced ESSO that reduces the overall carbon footprint of a telecommunication network by intelligently embedding and migrating SFCs across different locations. In addition, ESSO consolidates recently freed resources to reduce energy consumption. To achieve these objectives, we identified the Multi-Location SFC Orchestration (MLSO) problem and presented an ILP formulation for it. The MLSO problem is $\mathcal{NP}$-hard; therefore, we devised a heuristic algorithm for ESSO. The fundamental idea of the heuristic is to generate a near-optimal SFC embedding across the infrastructure by piecing together partial embedding data collected from a small number of NFV-PoDs. We performed extensive simulations on four ISP topologies collected from the Rocketfuel dataset. Our results showed that the proposed heuristic algorithm provides near-optimal solutions within milliseconds for all topologies.

In this dissertation, we have addressed three key research challenges in network softwarization and proposed efficient solutions to solve them. We have also demonstrated the superiority of our proposed solutions through extensive simulations using realistic scenarios. However, there are issues that need further research. The next section presents these future research directions.

5.2 Future Research Directions

Dynamic Controller Provisioning

The two heuristics for dynamic controller provisioning presented in this thesis demonstrate different performance characteristics. The DCP-GK heuristic is faster than DCP-SA, but DCP-SA provides better solutions with lower flow-setup time. It would be interesting to combine the two heuristics where DCP-GK generates initial solutions that are improved by DCP-SA. Moreover, a spike in control traffic is not handled properly in the current approach. The controller management system could reserve a certain amount of CPU capacity at each controller to handle traffic spikes. This mechanism can reduce the number of unnecessary switch-to-controller reassignment after a traffic spike. Furthermore, each

switch can be assigned one master and one or more slave controller(s) to guarantee fault-tolerance of the control plane. Another potential research direction is to use predictive analysis of the `packet_in` messages at different controllers to predict future controller load and adapt the active controller pool accordingly.

Orchestrating Virtual Network Functions

The work on VNF orchestration can be extended in a number of ways. First, the VNF orchestration model can be extended to support both hardware and software middleboxes for embedding different functions from the same service chain. For example, network functions that require a large number of cryptographic operations might gain by running on top of custom hardware middleboxes. The extended model will be able to support such use cases. Moreover, failure-resilient chain embedding by deploying backup VNFs can be investigated. In this case, each service chain will be embedded with a primary path and backup path(s) that will be utilized when a VNF on the primary path fails. Furthermore, the backup paths can be shared between multiple service chains to reduce the amount of underutilized resources in the network. Another interesting research direction is to use a testbed to measure the actual power consumption by the different components of a service chain to gain a deeper understanding about the interaction between different types of VNFs and network traffic.

Energy Smart Service Function Chain Orchestration (ESSO)

In ESSO, service chain migration decisions are taken based on the migration threshold, which can be extended by modeling the impact of migration. Service downtime can be used as a potential parameter for the extended model. In addition, the accumulated downtime experienced by a service in its lifetime should be considered while making migration decisions. The accumulated downtime must not exceed the maximum allowed downtime agreed in the SLA. Moreover, alternate cost models that consider factors other than carbon footprint can be investigated. Another interesting avenue of research will be to explore a pricing model for each service chain. In this case, the operator will profit from embedding a service chain but pay for electricity and penalties for carbon emission and SLA violations. Furthermore, the current model for estimating renewable energy can be extended to consider other green energy sources, *e.g.*, a network operator might purchase Renewable Energy Certificates (RECs) from green energy vendors while still consuming energy from a conventional brown energy grid. In this case, the problem becomes more interesting as REC prices vary based on location.

References

[1] K. G. Coffman and A. M. Odlyzko, "Growth of the Internet," *Optical Fiber Telecommunications IV B: Systems and Impairments, IP Kaminow and T. Li, eds*, pp. 17–56, 2002.

[2] "Cisco Visual Networking Index: Forecast and Methodology, 2016–2021 - Cisco," https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/complete-white-paper-c11-481360.html.

[3] J. Donovan and K. Prabhu, *Building the Network of the Future: Getting Smarter, Faster, and More Flexible with a Software Centric Approach*. CRC Press, 2017.

[4] L. Atzori, A. Iera, and G. Morabito, "The Internet of Things: A survey," *Computer Networks*, vol. 54, no. 15, pp. 2787–2805, 2010. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1389128610001568

[5] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu *et al.*, "B4: Experience with a globally-deployed software defined wan," in *ACM SIGCOMM Computer Communication Review (CCR)*, vol. 43, no. 4. ACM, 2013, pp. 3–14.

[6] F. De Turck, J.-M. Kang, H. Choo, M.-S. Kim, B.-Y. Choi, R. Badonnel, and J. W.-K. Hong, "Softwarization of networks, clouds, and internet of things," *International Journal of Network Management*, vol. 27, no. 2, 2017.

[7] "OpenStack Open Source Cloud Computing Software," http://openstack.org/.

[8] "OpenDaylight (ODL) Platform," https://www.opendaylight.org/.

[9] "Open Platform for NFV (OPNFV)," https://www.opnfv.org/.

[10] D. Kreutz, F. M. V. Ramos, P. E. Veríssimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," *Proceedings of the IEEE*, vol. 103, no. 1, pp. 14–76, Jan 2015.

[11] R. Mijumbi, J. Serrat, J. L. Gorricho, N. Bouten, F. D. Turck, and R. Boutaba, "Network Function Virtualization: State-of-the-Art and Research Challenges," *IEEE Communications Surveys and Tutorials*, vol. 18, no. 1, pp. 236–262, 2016.

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

[13] V. Sekar, S. Ratnasamy, M. K. Reiter, N. Egi, and G. Shi, "The Middlebox Manifesto: Enabling Innovation in Middlebox Deployment," in *ACM SIGCOMM Workshop on Hot Topics in Networks (HotNets)*, 2011, pp. 1–6.

[14] J. Sherry, S. Hasan, C. Scott, A. Krishnamurthy, S. Ratnasamy, and V. Sekar, "Making middleboxes someone else's problem: network processing as a cloud service," *ACM SIGCOMM Computer Communication Review (CCR)*, vol. 42, no. 4, pp. 13–24, 2012.

[15] B. Heller, R. Sherwood, and N. McKeown, "The controller placement problem," in *ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking (HotSDN)*, 2012, pp. 7–12. [Online]. Available: http://doi.acm.org/10.1145/2342441.2342444

[16] A. Voellmy and J. Wang, "Scalable software defined network controllers," in *ACM SIGCOMM*. ACM, 2012, pp. 289–290.

[17] M. Yu, J. Rexford, M. J. Freedman, and J. Wang, "Scalable flow-based networking with DIFANE," in *ACM SIGCOMM*, 2012, pp. 351–362.

[18] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee, "DevoFlow: scaling flow management for high-performance networks," in *ACM SIGCOMM*, 2011, pp. 254–265.

[19] D. Levin, A. Wundsam, B. Heller, N. Handigol, and A. Feldmann, "Logically centralized?: state distribution trade-offs in software defined networks," in *ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking (HotSDN)*, 2012, pp. 1–6.

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

[21] T. Koponen, M. Casado, N. Gude, J. Stribling, L. Poutievski, M. Zhu, R. Ramanathan, Y. Iwata, H. Inoue, T. Hama *et al.*, "Onix: A distributed control platform for large-scale production networks," in *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, vol. 10. USENIX, 2010, pp. 1–6. [Online]. Available: http://dl.acm.org/citation.cfm?id=1924943.1924968

[22] S. Hassas Yeganeh and Y. Ganjali, "Kandoo: a framework for efficient and scalable offloading of control applications," in *ACM SIGCOMM Workshop on Hot Topics in Software Defined Networks (HotSDN)*. ACM, 2012, pp. 19–24.

[23] A. M. Carlos, C. E. Rothenberg, and F. M. Maurício, "In-packet Bloom filter based data center networking with distributed OpenFlow controllers," in *IEEE GLOBECOM Workshops (GC Wkshps)*. IEEE, 2010, pp. 584–588.

[24] A. Tootoonchian and Y. Ganjali, "HyperFlow: A distributed control plane for OpenFlow," in *Internet Network Management Conference on Research on Enterprise Networking (INM/WREN)*, 2010, pp. 3–3.

[25] A. S.-W. Tam, K. Xi, and H. J. Chao, "Use of devolved controllers in data center networks," in *IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. IEEE, 2011, pp. 596–601.

[26] O. Salman, I. H. Elhajj, A. Kayssi, and A. Chehab, "SDN controllers: A comparative study," in *Mediterranean Electrotechnical Conference (MELECON)*. IEEE, 2016, pp. 1–6.

[27] B. Xiong, K. Yang, J. Zhao, W. Li, and K. Li, "Performance Evaluation of OpenFlow-based Software-defined Networks Based on Queueing Model," *Computer Networks*, vol. 102, no. C, pp. 172–185, Jun. 2016. [Online]. Available: https://doi.org/10.1016/j.comnet.2016.03.005

[28] A. Tootoonchian, S. Gorbunov, Y. Ganjali, M. Casado, and R. Sherwood, "On Controller Performance in Software-defined Networks," in *USENIX Conference on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services (Hot-ICE)*, ser. Hot-ICE'12. USENIX, 2012, pp. 10–10. [Online]. Available: http://dl.acm.org/citation.cfm?id=2228283.2228297

[29] D. Pariag and T. Brecht, "Application Bandwidth and Flow Rates from 3 Trillion Flows Across 45 Carrier Networks," in *International Conference on Passive and Active Network Measurement (PAM)*. Springer, 2017, pp. 129–141.

[30] H. Wang, F. Xu, Y. Li, P. Zhang, and D. Jin, "Understanding mobile traffic patterns of large scale cellular towers in urban environment," in *ACM SIGCOMM Internet Measurement Conference (IMC)*. ACM, 2015, pp. 225–238.

[31] S. Kandula, S. Sengupta, A. Greenberg, P. Patel, and R. Chaiken, "The nature of data center traffic: measurements & analysis," in *ACM SIGCOMM Internet Measurement Conference (IMC)*. ACM, 2009, pp. 202–208.

[32] S. Huang, F. Cuadrado, and S. Uhlig, "Middleboxes in the Internet: A HTTP perspective," in *Network Traffic Measurement and Analysis Conference (TMA)*, June 2017, pp. 1–9.

[33] D. A. Joseph, A. Tavakoli, and I. Stoica, "A Policy-aware Switching Layer for Data Centers," in *ACM SIGCOMM*, 2008, pp. 51–62.

[34] J. Sherry and S. Ratnasamy, "A Survey of Enterprise Middlebox Deployments," EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2012-24, Feb 2012.

[35] C. Dixon, A. Krishnamurthy, and T. Anderson, "An end to the middle," in *Workshop on Hot Topics in Operating Systems (HotOS)*. USENIX Association, 2009, pp. 2–2.

[36] R. Mijumbi, J. Serrat, J. l. Gorricho, S. Latre, M. Charalambides, and D. Lopez, "Management and orchestration challenges in network functions virtualization," *IEEE Communications Magazine*, vol. 54, no. 1, pp. 98–105, January 2016.

[37] Z. A. Qazi, C.-C. Tu, L. Chiang, R. Miao, V. Sekar, and M. Yu, "SIMPLE-fying middlebox policy enforcement using SDN," in *ACM SIGCOMM*, 2013, pp. 27–38.

[38] P. Quinn and T. Nadeau, "Service Function Chaining Problem Statement," *draft-quinn-sfc-problem-statement-10 (work in progress)*, 2014.

[39] C. Xie, Q. Sun, W. Meng, C. Wang, and B. Khasnabish, "Service Function Chaining Use Cases in Broadband," *draft-meng-sfc-broadband-usecases-03 (work in progress)*, 2015.

[40] W. Haeffner, J. Napper, M. Stiemerling, D. Lopez, and J. Uttaro, "Service Function Chaining Use Cases in Mobile Networks," *draft-ietf-sfc-use-case-mobility-08 (work in progress)*, 2018.

[41] S. Surendra, M. Tufail, S. Majee, C. Captari, and S. Homma, "Service Function Chaining Use Cases in Data Centers," *draft-ietf-sfc-dc-case-06 (work in progress)*, 2017.

[42] J. Quittek, P. Bauskar, T. BenMeriem, A. Bennett, M. Besson, and A. Et, "Network Functions Virtualisation (NFV) - Management and Orchestration," *ETSI NFV ISG, White Paper*, 2014.

[43] F. Bari, S. R. Chowdhury, R. Ahmed, R. Boutaba, and O. C. M. B. Duarte, "Orchestrating Virtualized Network Functions," *IEEE Transactions on Network and Service Management*, vol. 13, no. 4, pp. 725–739, Dec 2016.

[44] ETSI, "Network Functions Virtualisation (NFV): Architectural Framework," ETSI GS NFV, Tech. Rep., 2013.

[45] J. Martins, M. Ahmed, C. Raiciu, V. Olteanu, M. Honda, R. Bifulco, and F. Huici, "ClickOS and the art of network function virtualization," in *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*. USENIX, 2014, pp. 459–473.

[46] J. Hwang, K. K. Ramakrishnan, and T. Wood, "NetVM: High performance and flexible networking using virtualization on commodity platforms," in *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*. USENIX, 2014, pp. 445–458.

[47] Sandvine, "Breaking the Terabit Barrier: A One Terabit per Second Policy Control Virtual Network Function," *Technology Showcase, Sandvine*, 2015. [Online]. Available: https://www.sandvine.com/downloads/general/sandvine-technology-showcases/breaking-the-terabit-barrier.pdf

[48] "G.W.A.T.T. – Global "What if" Analyzer of neTwork energy consumpTion," http://gwatt.net/.

[49] T. Joyce, T. A. Okrasinski, and W. Schaeffer, "Estimating the carbon footprint of telecommunications products: A heuristic approach," *Journal of Mechanical Design*, vol. 132, no. 9, pp. 094 502–094 502–4, 2010.

[50] H. S. Dunn, "The carbon footprint of ICTs," *Global Information Society Watch*, 2010.

[51] "Federal Renewable Energy Projects And Technologies," http://energy.gov/eere/femp/federal-renewable-energy-projects-and-technologies.

[52] P. Natsu, "53% Of Consumers Prefer To Buy From Company With Green Rep," http://www.environmentalleader.com/2007/10/02/53-of-consumers-prefer-to-buy-from-companies-with-green-rep/, 2007.

[53] "Carbon Tax Center," http://www.carbontax.org/.

[54] KPMG, "Carbon footprint stomps on firm value," https://goo.gl/r0B9fj.

[55] H. Ikebe, N. Yamashita, and R. Nishii, "Green energy for telecommunications," in *IEEE International Telecommunications Energy Conference (INTELEC)*. IEEE, 2007, pp. 750–755.

[56] D. Kumar and S. Shekhar, "Renewable Energy Potential for Green Networks," *Research Journal of Science and Technology*, vol. 7, no. 2, p. 125, 2015.

[57] R. Bolla, R. Bruschi, A. Carrega, F. Davoli, D. Suino, C. Vassilakis, and A. Zafeiropoulos, "Cutting the energy bills of Internet Service Providers and telecoms through power management: An impact analysis," *Computer Networks*, vol. 56, no. 10, pp. 2320–2342, 2012.

[58] T. Vu, V. Luc, N. Quan, T. Thanh, N. Thanh, and P. Nam, "Sleep Mode and Wakeup Method for OpenFlow Switches," *Journal of Low Power Electronics*, vol. 10, no. 3, pp. 347–353, 2014.

[59] S. Fang, H. Li, C. H. Foh, Y. Wen, and K. M. M. Aung, "Energy optimizations for data center network: Formulation and its solution," in *IEEE Global Communications Conference (GLOBECOM)*. IEEE, 2012, pp. 3256–3261.

[60] "USCRN/USRCRN Quality Controlled Datasets," https://www.ncdc.noaa.gov/crn/qcdatasets.html.

[61] A. Gember-Jacobson, R. Viswanathan, C. Prakash, R. Grandl, J. Khalid, S. Das, and A. Akella, "OpenNF: Enabling innovation in network function control," in *ACM SIGCOMM*. ACM, 2014, pp. 163–174.

[62] Y. Wang, G. Xie, Z. Li, P. He, and K. Salamatian, "Transparent flow migration for NFV," in *IEEE International Conference on Network Protocols (ICNP)*, Nov 2016, pp. 1–10.

[63] S. Rajagopalan, D. Williams, H. Jamjoom, and A. Warfield, "Split/merge: System support for elastic execution in virtual middleboxes." in *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2013, pp. 227–240.

[64] S. Palkar, C. Lan, S. Han, K. Jang, A. Panda, S. Ratnasamy, L. Rizzo, and S. Shenker, "E2: a framework for NFV applications," in *ACM Symposium on Operating Systems Principles (SOSP)*. ACM, 2015, pp. 121–136.

[65] "Open Networking Foundation (ONF)," https://www.opennetworking.org/.

[66] "Internet2 Research Network Topology and Traffic Matrix," http://www.cs.utexas.edu/~yzhang/research/AbileneTM/.

[67] "NOX," https://github.com/noxrepo/nox.

[68] J. Stribling, Y. Sovran, I. Zhang, X. Pretzer, J. Li, M. F. Kaashoek, and R. Morris, "Flexible, Wide-area Storage for Distributed Systems with WheelFS," in *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, ser. NSDI'09. Berkeley, CA, USA: USENIX Association, 2009, pp. 43–58. [Online]. Available: http://dl.acm.org/citation.cfm?id=1558977.1558981

[69] M. F. Bari, A. R. Roy, S. R. Chowdhury, Q. Zhang, M. F. Zhani, R. Ahmed, and R. Boutaba, "Dynamic Controller Provisioning in Software Defined Networks," in *International Conference on Network and Service Management (CNSM)*, Oct 2013, pp. 18–25.

[70] G. Yao, J. Bi, Y. Li, and L. Guo, "On the Capacitated Controller Placement Problem in Software Defined Networks," *IEEE Communications Letters*, vol. 18, no. 8, pp. 1339–1342, Aug 2014.

[71] D. Hock, M. Hartmann, S. Gebert, M. Jarschel, T. Zinner, and P. Tran-Gia, "Pareto-optimal resilient controller placement in SDN-based core networks," in *International Teletraffic Congress (ITC)*, Sept 2013, pp. 1–9.

[72] D. Hock, S. Gebert, M. Hartmann, T. Zinner, and P. Tran-Gia, "POCO-framework for Pareto-optimal resilient controller placement in SDN-based core networks," in *IEEE Network Operations and Management Symposium (NOMS)*, May 2014, pp. 1–2.

[73] M. Obadia, M. Bouet, J. L. Rougier, and L. Iannone, "A greedy approach for minimizing SDN control overhead," in *IEEE Conference on Network Softwarization (NetSoft)*, April 2015, pp. 1–5.

[74] Y. Hu, W. Wang, X. Gong, X. Que, and S. Cheng, "On reliability-optimized controller placement for Software-Defined Networks," *China Communications*, vol. 11, no. 2, pp. 38–54, Feb 2014.

[75] Y. Hu, W. Wendong, X. Gong, X. Que, and C. Shiduan, "Reliability-aware controller placement for Software-Defined Networks," in *IFIP/IEEE International Symposium on Integrated Network Management (IM)*, May 2013, pp. 672–675.

[76] F. J. Ros and P. M. Ruiz, "Five Nines of Southbound Reliability in Software-defined Networks," in *ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking (HotSDN)*. New York, NY, USA: ACM, 2014, pp. 31–36. [Online]. Available: http://doi.acm.org/10.1145/2620728.2620752

[77] Y. Jiménez, C. Cervelló-Pastor, and A. J. García, "On the controller placement for designing a distributed SDN control layer," in *IFIP Networking Conference*, June 2014, pp. 1–9.

[78] D. M. F. Mattos, O. C. M. B. Duarte, and G. Pujolle, "A resilient distributed controller for software defined networking," in *IEEE International Conference on Communications (ICC)*, May 2016, pp. 1–6.

[79] P. Patil, A. Gokhale, and A. Hakiri, "Bootstrapping Software Defined Network for flexible and dynamic control plane management," in *IEEE Conference on Network Softwarization (NetSoft)*. IEEE, 2015, pp. 1–5.

[80] A. Raza, A. Gohar, and S. Lee, "MPTCP based in-band controlling for the software defined networks," in *International Conference on Information and Communication Technology Convergence (ICTC)*. IEEE, 2017, pp. 163–167.

[81] M. Canini, I. Salem, L. Schiff, E. M. Schiller, and S. Schmid, "A self-organizing distributed and in-band SDN control plane," in *IEEE International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2017, pp. 2656–2657.

[82] M. Cello, Y. Xu, A. Walid, G. Wilfong, H. J. Chao, and M. Marchese, "BalCon: A Distributed Elastic SDN Control via Efficient Switch Migration," in *IEEE International Conference on Cloud Engineering (IC2E)*. IEEE, 2017, pp. 40–50.

[83] A. Dixit, F. Hao, S. Mukherjee, T. Lakshman, and R. R. Kompella, "ElastiCon; an elastic distributed SDN controller," in *ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*. IEEE, 2014, pp. 17–27.

[84] A. Basta, A. Blenk, H. B. Hassine, and W. Kellerer, "Towards a dynamic SDN virtualization layer: Control path migration protocol," in *IEEE International Conference on Network and Service Management (CNSM)*. IEEE, 2015, pp. 354–359.

[85] O. N. Foundation, "OpenFlow switch specification 1.5.1," Open Networking Foundation, Tech. Rep., Mar. 2015.

[86] P. Berde, M. Gerola, J. Hart, Y. Higuchi, M. Kobayashi, T. Koide, B. Lantz, B. O'Connor, P. Radoslavov, W. Snow *et al.*, "ONOS: towards an open, distributed SDN OS," in *ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking (HotSDN)*. ACM, 2014, pp. 1–6.

[87] H. Yin, H. Xie, T. Tsou, D. Lopez, P. Aranda, and R. Sidi, "SDNi: A Message Exchange Protocol for Software Defined Networks (SDNS) across Multiple Domains," Internet Draft, Internet Engineering Task Force, June 2012. [Online]. Available: http://tools.ietf.org/id/draft-yin-sdn-sdni-00.txt

[88] P. Lin, J. Bi, and Y. Wang, "East-west bridge for sdn network peering," in *Frontiers in Internet Technologies*. Springer, 2013, pp. 170–181.

[89] Z. Cai, A. L. Cox, and T. S. E. Ng, "Maestro: A System for Scalable OpenFlow Control," Rice University, Tech. Rep., 2011.

[90] S. G. Kolliopoulos and C. Stein, "Improved approximation algorithms for unsplittable flow problems," in *FOCS*, 1997, pp. 426–436. [Online]. Available: http://dl.acm.org/citation.cfm?id=795663.796365

[91] N. Spring, R. Mahajan, and D. Wetherall, "Measuring ISP topologies with rocketfuel," in *ACM SIGCOMM*, 2002, pp. 133–145.

[92] S. Gebert, R. Pries, D. Schlosser, and K. Heck, "Internet Access Traffic Measurement and Analysis," in *Traffic Monitoring and Analysis*, ser. LNCS, 2012, vol. 7189, pp. 29–42.

[93] "Enhanced Network Survivability Performance," Ansi t1.tr.68-2001, 2011.

[94] W. Liu, H. Li, O. Huang, M. Boucadair, N. Leymann, Q. Fu, Q. Sun, C. Pham, C. Huang, J. Zhu, and P. He, "Service Function Chaining (SFC) General Use Cases," *draft-liu-sfc-use-cases-08 (work in progress)*, 2014.

[95] S. Surendra, M. Tufail, S. Majee, C. Captari, and S. Homma, "Service Function Chaining Use Cases in Data Centers," *draft-ietf-sfc-dc-case-06 (work in progress)*, 2017.

[96] M. F. Bari, R. Boutaba, R. P. Esteves, L. Z. Granville, M. Podlesny, M. G. Rabbani, Q. Zhang, and M. F. Zhani, "Data Center Network Virtualization: A Survey," *IEEE Communications Surveys and Tutorials*, vol. 15, no. 2, pp. 909–928, 2013.

[97] P. L. D. Puglia and G. Francesca, "A Survey of Resource Constrained Shortest Path Problems: Exact Solution Approaches," *Networks*, vol. 62, no. 3, pp. 183–200, 2013.

[98] Q. Zhang, M. F. Zhani, R. Boutaba, and J. L. Hellerstein, "HARMONY: Dynamic Heterogeneity-Aware Resource Provisioning in the Cloud," in *IEEE International Conference on Distributed Computing Systems (ICDCS)*, July 2013, pp. 510–519.

[99] Q. Zhang, Q. Zhu, M. F. Zhani, R. Boutaba, and J. L. Hellerstein, "Dynamic Service Placement in Geographically Distributed Clouds," *IEEE Journal on Selected Areas in Communications*, vol. 31, no. 12, pp. 762–772, December 2013.

[100] M. C. Luizelli, L. R. Bays, L. S. Buriol, M. P. Barcellos, and L. P. Gaspary, "Piecing Together the NFV Provisioning Puzzle: Efficient Placement and Chaining of Virtual Network Functions," in *IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, May 2015, pp. 98–106.

[101] ETSI, "Network Functions Virtualisation – Introductory White Paper," https://portal.etsi.org/NFV/NFV_White_Paper.pdf, 2012.

[102] "Comparison of enterprise class power enclosure," http://www.dell.com/downloads/global/products/pedge/en/bladepower-studywhitepaper_08112010_final.pdf.

[103] "Internet Transit Pricing," http://drpeering.net/white-papers/Internet-Transit-Pricing-Historical-And-Projected.php.

[104] N. M. K. Chowdhury and R. Boutaba, "A survey of network virtualization," *Computer Networks*, vol. 54, no. 5, pp. 862–876, 2010.

[105] L. T. Kou and G. Markowsky, "Multidimensional bin packing algorithms," *IBM Journal of Research and Development*, vol. 21, no. 5, pp. 443–448, 1977.

[106] R. Sridharan, "A lagrangian heuristic for the capacitated plant location problem with single source constraints," *European Journal of Operational Research*, vol. 66, no. 3, pp. 305–312, 1993.

[107] G. D. Forney Jr, "The Viterbi Algorithm," *Proceedings of the IEEE*, vol. 61, no. 3, pp. 268–278, 1973.

[108] T. Benson, A. Akella, and D. A. Maltz, "Network traffic characteristics of data centers in the wild," in *ACM SIGCOMM Internet Measurement Conference (IMC)*, 2010, pp. 267–280.

[109] N. Spring, R. Mahajan, and D. Wetherall, "Measuring ISP topologies with Rocketfuel," *ACM SIGCOMM Computer Communication Review (CCR)*, vol. 32, no. 4, pp. 133–145, 2002.

[110] L. Saino, C. Cocora, and G. Pavlou, "A Toolchain for Simplifying Network Simulation Setup," in *EAI International Conference on Simulation Tools and Techniques*, 2013.

[111] A. Nucci, A. Sridharan, and N. Taft, "The problem of synthetically generating ip traffic matrices: initial recommendations," *ACM SIGCOMM Computer Communication Review (CCR)*, vol. 35, no. 3, pp. 19–32, 2005.

[112] "https://datatracker.ietf.org/doc/draft-ietf-sfc-dc-use-cases/."

[113] "pfSense Hardware Sizing Guide," https://www.pfsense.org/hardware/#sizing.

[114] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," *ACM SIGCOMM Computer Communication Review (CCR)*, vol. 38, no. 4, pp. 63–74, Aug 2008. [Online]. Available: http://doi.acm.org/10.1145/1402946.1402967

[115] G. Gibb, H. Zeng, and N. McKeown, "Outsourcing network functionality," in *ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking (HotSDN)*, 2012, pp. 73–78.

[116] A. Gember, A. Krishnamurthy, S. S. John, R. Grandl, X. Gao, A. Anand, T. Benson, A. Akella, and V. Sekar, "Stratos: A network-aware orchestration layer for middleboxes in the cloud," *CoRR*, vol. abs/1305.0209, 2013. [Online]. Available: http://arxiv.org/abs/1305.0209

[117] S. K. Fayazbakhsh, L. Chiang, V. Sekar, M. Yu, and J. C. Mogul, "Enforcing network-wide policies in the presence of dynamic middlebox actions using flowtags," in *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*. USENIX, 2014, pp. 533–546. [Online]. Available: http://dl.acm.org/citation.cfm?id=2616448.2616497

[118] S. Mehraghdam, M. Keller, and H. Karl, "Specifying and Placing Chains of Virtual Network Functions," in *IEEE International Conference on Cloud Networking (CloudNet)*, Oct 2014, pp. 7–13.

[119] R. Cohen, L. Lewin-Eytan, J. Naor, and D. Raz, "Near optimal placement of virtual network functions," in *IEEE International Conference on Computer Communication (INFOCOM)*, 2015, pp. 1346–1354.

[120] W. Rankothge, J. Ma, F. Le, A. Russo, and J. Lobo, "Towards making network function virtualization a cloud computing service," in *IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, May 2015, pp. 89–97.

[121] S. Clayman, E. Maini, A. Galis, A. Manzalini, and N. Mazzocca, "The dynamic placement of virtual network functions," in *IFIP/IEEE Network Operations and Management Symposium (NOMS)*, 2014, pp. 1–9.

[122] M. Mechtri, C. Ghribi, and D. Zeghlache, "A Scalable Algorithm for the Placement of Service Function Chains," *IEEE Transactions on Network and Service Management*, vol. 13, no. 3, pp. 533–546, Sept 2016.

[123] X. Li and C. Qian, "The virtual network function placement problem," in *IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, April 2015, pp. 69–70.

[124] T. Lin, Z. Zhou, M. Tornatore, and B. Mukherjee, "Optimal Network Function Virtualization Realizing End-to-End Requests," in *IEEE Global Communications Conference (GLOBECOM)*, Dec 2015, pp. 1–6.

[125] B. Addis, D. Belabed, M. Bouet, and S. Secci, "Virtual network functions placement and routing optimization," in *IEEE International Conference on Cloud Networking (CloudNet)*, Oct 2015, pp. 171–177.

[126] C. Pham, N. H. Tran, S. Ren, W. Saad, and C. S. Hong, "Traffic-aware and Energy-efficient vNF Placement for Service Chaining: Joint Sampling and Matching Approach," *IEEE Transactions on Services Computing*, pp. 1–1, 2017.

[127] W. Ma, O. Sandoval, J. Beltran, D. Pan, and N. Pissinou, "Traffic aware placement of interdependent NFV middleboxes," in *IEEE International Conference on Computer Communications (INFOCOM)*, May 2017, pp. 1–9.

[128] "Intel DPDK," http://dpdk.org/.

[129] L. Rizzo, "netmap: A Novel Framework for Fast Packet I/O." in *USENIX Annual Technical Conference (ATC)*, 2012, pp. 101–112.

[130] V. Sekar, N. Egi, S. Ratnasamy, M. K. Reiter, and G. Shi, "Design and Implementation of a Consolidated Middlebox Architecture." in *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2012, pp. 323–336.

[131] J. W. Anderson, R. Braud, R. Kapoor, G. Porter, and A. Vahdat, "xOMB: Extensible open middleboxes with commodity servers," in *ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*, 2012, pp. 49–60.

[132] L. Peterson, A. Al-Shabibi, T. Anshutz, S. Baker, A. Bavier, S. Das, J. Hart, G. Palukar, and W. Snow, "Central office re-architected as a data center," *IEEE Communications Magazine*, vol. 54, no. 10, pp. 96–101, 2016.

[133] "100% renewable is just the beginning," https://environment.google/projects/announcement-100/.

[134] "Virtual Central Office – OPNFV," https://www.opnfv.org/wp-content/uploads/sites/12/2017/09/OPNFV_VCO_Oct17.pdf.

[135] J. Halpern and C. Pignataro, "Service function chaining (sfc) architecture," Internet Engineering Task Force (IETF), Tech. Rep., 2015.

[136] H. Freeman and R. Boutaba, "Networking ind ustry transf ormation through softwarization [the president's page]," *IEEE Communications Magazine*, vol. 54, no. 8, pp. 4–6, August 2016.

[137] D. Kreutz, F. M. Ramos, P. E. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," *Proceedings of the IEEE*, vol. 103, no. 1, pp. 14–76, 2015.

[138] "How to calculate power output of wind," http://www.windpowerengineering.com/construction/calculate-wind-power-output/.

[139] "Energy Management," http://about.att.com/content/csr/home/ issue-brief-builder/environment/energy-management.html.

[140] "AT&T 2025 Goals," http://www.about.att.com/csr/goals.

[141] "State Electricity Profiles," https://www.eia.gov/electricity/state/.

[142] B. Heller, S. Seetharaman, P. Mahadevan, Y. Yiakoumis, P. Sharma, S. Banerjee, and N. McKeown, "ElasticTree: Saving Energy in Data Center Networks," in *USENIX Conference on Networked Systems Design and Implementation (NSDI)*, vol. 10, 2010, pp. 249–264.

[143] "PowerConnect 8024F Switch," http://www.dell.com/us/ business/p/powerconnect-8024f/pd.

[144] T. H. Vu, T. Thanh, V. Q. Trong, P. N. Nam, and N. H. Thanh, "NetFPGA Based OpenFlow Switch Extension for Energy Saving in Data Centers," *Journal on Electronics and Communications*, vol. 3, no. 1–2, 2013.

[145] "CORD: Reinventing Central Offices For Efficiency & Agility," http://opencord.org/.

[146] A. Al-Shabibi and L. Peterson, "CORD: Central Office Re-architected as a Datacenter," Open Stack Summit, Tech. Rep., 2015.

[147] M. Kablan, A. Alsudais, E. Keller, and F. Le, "Stateless Network Functions: Breaking the Tight Coupling of State and Processing," in *USENIX Conference on Networked Systems Design and Implementation (NSDI)*, ser. NSDI'17. Berkeley, CA, USA: USENIX Association, 2017, pp. 97–112. [Online]. Available: http://dl.acm.org/citation.cfm?id=3154630.3154639

[148] "AT&T Domain 2.0 Vision White Paper," https://www.att.com/Common/about_us/ pdf/AT&T Domain 2.0 Vision White Paper.pdf.

[149] Y. Xie, Z. Liu, S. Wang, and Y. Wang, "Service Function Chaining Resource Allocation: A Survey," *eprint arXiv:1608.00095*, 2016. [Online]. Available: http://arxiv.org/abs/1608.00095

[150] J. A. Aroca, A. Chatzipapas, A. F. Anta, and V. Mancuso, "A Measurement-Based Characterization of the Energy Consumption in Data Center Servers," *IEEE Journal on Selected Areas in Communications (JSAC)*, vol. 33, no. 12, pp. 2863–2877, Dec 2015.

[151] "Bin Packing Problem," https://en.wikipedia.org/wiki/ Bin_packing_problem.

[152] R. E. Korf, "A New Algorithm for Optimal Bin Packing," in *Eighteenth National Conference on Artificial Intelligence*. American Association for Artificial Intelligence, 2002, pp. 731–736. [Online]. Available: http://dl.acm.org/citation.cfm?id=777092.777205

[153] X. Li and C. Qian, "An nfv orchestration framework for interference-free policy enforcement," in *IEEE International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2016, pp. 649–658.

[154] Q. Zhang, Y. Xiao, F. Liu, J. C. Lui, J. Guo, and T. Wang, "Joint optimization of chain placement and request scheduling for network function virtualization," in *IEEE International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2017, pp. 731–741.

[155] S. Ayoubi, S. R. Chowdhury, and R. Boutaba, "Breaking service function chains with khaleesi," in *IFIP Networking Conference*, 2018.

[156] S. Kumar, M. Tufail, S. Majee, C. Captari, and S. Homma, "Service function chaining use cases in data centers," *IETF SFC WG*, 2015.

[157] A. Nucci, A. Sridharan, and N. Taft, "The Problem of Synthetically Generating IP Traffic Matrices: Initial Recommendations," *ACM SIGCOMM Computer Communication Review (CCR)*, vol. 35, no. 3, pp. 19–32, Jul. 2005. [Online]. Available: http://doi.acm.org/10.1145/1070873.1070876

[158] D. Meisner and T. F. Wenisch, "Peak power modeling for data center servers with switched-mode power supplies," *ACM/IEEE International Symposium on Low-Power Electronics and Design (ISLPED)*, pp. 319–324, 2010.

[159] "CPLEX Optimizer," https://www.ibm.com/analytics/cplex-optimizer.

[160] V. Eramo, E. Miucci, M. Ammar, and F. G. Lavacca, "An Approach for Service Function Chain Routing and Virtual Function Network Instance Migration in Network Function Virtualization Architectures," *IEEE/ACM Transactions on Networking*, vol. 25, no. 4, pp. 2008–2025, Aug 2017.

[161] M. F. Bari, S. R. Chowdhury, R. Ahmed, and R. Boutaba, "On orchestrating virtual network functions," in *International Conference on Network and Service Management (CNSM)*, ser. CNSM '15. Washington, DC, USA: IEEE Computer Society, 2015, pp. 50–56. [Online]. Available: http://dx.doi.org/10.1109/CNSM.2015.7367338

[162] C. Ghribi, M. Mechtri, and D. Zeghlache, "A Dynamic Programming Algorithm for Joint VNF Placement and Chaining," in *ACM Workshop on Cloud-Assisted Networking (CAN)*, ser. CAN '16. New York, NY, USA: ACM, 2016, pp. 19–24. [Online]. Available: http://doi.acm.org/10.1145/3010079.3010083

[163] M. F. Bari, S. R. Chowdhury, R. Ahmed, and R. Boutaba, "On Orchestrating Virtual Network Functions in NFV," *Computing Research Repository (CoRR)*, vol. abs/1503.06377, 2015. [Online]. Available: http://arxiv.org/abs/1503.06377

[164] X. Guan, B.-Y. Choi, and S. Song, "Energy Efficient Virtual Network Embedding for Green Data Centers Using Data Center Topology and Future Migration," *Computer Communications*, vol. 69, no. C, pp. 50–59, Sep. 2015. [Online]. Available: https://doi.org/10.1016/j.comcom.2015.05.003

[165] P. X. Gao, A. R. Curtis, B. Wong, and S. Keshav, "It's not easy being green," in *ACM SIGCOMM*. ACM, 2012, pp. 211–222.

[166] A. Amokrane, R. Langar, M. Zhani, R. Boutaba, and G. Pujolle, "Greenslater: On Satisfying Green SLAs in Distributed Clouds," *IEEE Transactions on Network and Service Management (TNSM)*, vol. 12, no. 3, pp. 363–376, Sept 2015.

[167] A. Amokrane, M. F. Zhani, R. Langar, R. Boutaba, and G. Pujolle, "Greenhead: Virtual data center embedding across distributed infrastructures," *IEEE Transactions on Cloud Computing (TCC)*, vol. 1, no. 1, pp. 36–49, 2013.

[168] A. Amokrane, M. F. Zhani, Q. Zhang, R. Langar, R. Boutaba, and G. Pujolle, "On Satisfying Green SLAs in Distributed Clouds," in *IEEE/ACM/IFIP International Conference on Network and Service Management (CNSM)*, Rio de Janeiro, Brazil, November 2014, pp. 64–72.