Resource Management in Softwarized Networks

by

Shihabur Rahman Chowdhury

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, 2021

© Shihabur Rahman Chowdhury 2021

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: Mostafa Ammar
Regents' Professor
School of Computer Science
Georgia Institute of Technology

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: Ravi Mazumdar
Professor
Department of Electrical and Computer Engineering
University of Waterloo

Author's Declaration

This thesis consists of material all of which I authored or co-authored: see Statement of Contributions included in the 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.

Statement of Contributions

This dissertation includes first authored peer-reviewed materials that have appeared in journals and conference proceedings published by the Institute of Electrical and Electronics Engineers (IEEE). The IEEE's policy on reuse of published materials in a dissertation is as follows: "*The IEEE does not require individuals working on a thesis to obtain a formal reuse license*".

The following list serves as a declaration of the Versions of Record for works included in this dissertation:

Portions of Chapter 2:

S. R. Chowdhury, M. A. Salahuddin, N. Limam, and R. Boutaba. "Re-architecting NFV Ecosystem with Microservices: State-of-the-art and Research Challenges", in *IEEE Network*, vol. 33, no. 3, pp. 168-176, May/June 2019, doi: 10.1109/MNET.2019.1800082.

S. R. Chowdhury, Anthony, H. Bian, T. Bai, and R. Boutaba. "μNF: A Disaggregated Packet Processing Architecture". in *IEEE Conference on Network Softwarization (NetSoft)*, 2019, pp. 342-350, doi: 10.1109/NETSOFT.2019.8806657.

S. R. Chowdhury, Anthony, H. Bian, T. Bai, and R. Boutaba. "A Disaggregated Packet Processing Architecture for Network Function Virtualization". in *IEEE Journal on Selected Areas in Communications*, vol. 38, no. 6, pp. 1075-1088, June 2020, doi: 10.1109/JSAC.2020.2986611.

Portions of Chapter 3:

S. R. Chowdhury, S. Ayoubi, R. Ahmed, N. Shahriar, R. Boutaba, J. Mitra, and L. Liu. "MULE: Multi-Layer Virtual Network Embedding", in *IEEE/ACM/IFIP International Conference on Network and Service Management (CNSM)*, 2017, pp. 1-9, doi: 10.23919/CNSM.2017.8256005.

S. R. Chowdhury, S. Ayoubi, R. Ahmed, N. Shahriar, R. Boutaba, J. Mitra, and L. Liu. "Multi-Layer Virtual Network Embedding", in *IEEE Transactions on Network and Service Management*, vol. 15, no. 3, pp. 1132-1145, September 2018, doi: 10.1109/TNSM.2018.2834315.

Portions of Chapter 4:

S. R. Chowdhury, R. Ahmed, M. M. A. Khan, N. Shahriar, R. Boutaba, J. Mitra, and F. Zeng. "Protecting virtual networks with DRONE", in *IEEE/IFIP Network Operations and Management Symposium (NOMS)*, 2016, pp. 78-86, doi: 10.1109/NOMS.2016.7502799.

S. R. Chowdhury, R. Ahmed, M. M. A. Khan, N. Shahriar, R. Boutaba, J. Mitra, and F. Zeng. "Dedicated Protection for Survivable Virtual Network Embedding", in *IEEE Transactions on Network and Service Management*, vol. 13, no. 4, pp. 913-926, December 2016, doi: 10.1109/TNSM.2016.2574239.

Portions of Chapter 5:

S. R. Chowdhury, M. F. Bari, R. Ahmed, and R. Boutaba, "PayLess: A low cost network monitoring framework for Software Defined Networks", in *IEEE/IFIP Network Operations and Management Symposium (NOMS)*, 2014, pp. 1-9, doi: 10.1109/NOMS.2014.6838227.

S. R. Chowdhury, R. Boutaba, and J. François, "LINT: Accuracy-adaptive and Lightweight In-band Network Telemetry", in *IFIP/IEEE International Symposium on Integrated Network Management (IM)*, 2021 (*to appear*).

Abstract

Communication networks are undergoing a major transformation through *softwarization*, which is changing the way networks are designed, operated, and managed. *Network Softwarization* is an emerging paradigm where software controls the treatment of network flows, adds value to these flows by software processing, and orchestrates the on-demand creation of customized networks to meet the needs of customer applications. *Software-Defined Networking (SDN)*, *Network Function Virtualization (NFV)*, and *Network Virtualization* are three cornerstones of the overall transformation trend toward network softwarization. Together, they are empowering network operators to accelerate time-to-market for new services, diversify the supply chain for networking hardware and software, bringing the benefits of agility, economies of scale, and flexibility of cloud computing to networks. The enhanced programmability enabled by softwarization creates unique opportunities for adapting network resources in support of applications and users with diverse requirements. To effectively leverage the flexibility provided by softwarization and realize its full potential, it is of paramount importance to devise proper mechanisms for allocating resources to different applications and users and for monitoring their usage over time.

The overarching goal of this dissertation is to advance state-of-the-art in how resources are allocated and monitored and build the foundation for effective resource management in softwarized networks. Specifically, we address four resource management challenges in three key enablers of network softwarization, namely SDN, NFV, and network virtualization. First, we challenge the current practice of realizing network services with monolithic software network functions and propose a microservice-based disaggregated architecture enabling finer-grained resource allocation and scaling. Then, we devise optimal solutions and scalable heuristics for establishing virtual networks with guaranteed bandwidth and guaranteed survivability against failure on multi-layer IP-over-Optical and single-layer IP substrate network, respectively. Finally, we propose adaptive sampling mechanisms for balancing the overhead of softwarized network monitoring and the accuracy of the network view constructed from monitoring data.

Acknowledgement

First of all, I thank God (Allah) for bringing me to existence, giving me the privilege of a comfortable life and the opportunity to pursue higher education at the best institutions, and enabling me to complete my work.

I wholeheartedly thank my supervisor Prof. Raouf Boutaba for his guidance, support, and encouragement over the years. He has taught me invaluable lessons in terms of not compromising the quality of work, and maintaining strong work ethics and professionalism. I hope to keep holding these high standards in the future. I am also thankful to him for always giving me the freedom and support for pursuing my own ideas. He was always available whether for brainstorming and technical discussion, constructively criticizing my work, helping me to improve my writing, and even one time for teaching me how to "drive a stick". He has not only been a mentor to me in academic research but also in ways of life. I hope to be able to turn to him for his mentorship in the years to come. I am also indebted to Prof. Raouf and Noura for their hospitality over the years, very often making it feel like home during special occasions.

I am grateful to Reaz and Faiz for their mentorship during the early days of graduate studies. Their guidance during the early days was utterly helpful in getting a grasp of the research process. Towards the end of my graduate studies, I gained significant exposure to optical networking through the collaboration with Prof. Massimo. My scholarly work is a culmination of many successful collaborations over the years. I was very fortunate to mentor and to work with Anthony, Haibo, and Tim while working on a disaggregated virtual network function architecture (Chapter 2). Most of my contributions in network virtualization was in collaboration with Huawei Technologies Canada Research Center, Ottawa. The industry partner's input to these works have greatly improved the motivation, practicality, and applicability of the research. I closely collaborated with Sara and Nashid for the multi-layer virtual network embedding (Chapter 3) and for the survivable virtual network embedding (Chapter 4) research, respectively. I am thankful to these excellent collaborators who made the work enjoyable. My decade long professional relationship with Nashid has resulted in many successful collaborations, also providing me the opportunity to take inspiration from his discipline and determination. Finally, my works on softwarized network monitoring (Chapter 5) were in close collaboration with Reaz and Faiz, and with Jérôme from INRIA. For the work on in-band network telemetry, the discussion with Prof. Samer was helpful in devising the proper implementation. Apart from the technical contributions, my colleagues at the networking lab and beyond, Faiz, Arup, Rabbani, Reaz, Nashid, Faten, Milad, Ben, Mashrur, Anthony, Haibo, Tim, Elahe, Eric, Sara, Felipe, and Amine were a great source of many interesting discussions related and unrelated to systems and networking.

I would like to thank my internship mentors Constantin, Valas, and Jérôme from IBM Research, Google, and INRIA, respectively, for their guidance during the respective internships.

Each of my internship experience was unique and rewarding. I had the opportunity to experience both doing cutting-edge research and translating ideas into production ready systems through these experiences.

I am thankful to my dissertation committee members: Professor Martin Karsten, Bernard Wong, Ravi Mazumdar and Mostafa Ammar, for their time in reviewing and providing valuable comments on this dissertation. I thoroughly enjoyed the conversation with them during the oral defense.

I am forever indebted to my parents Akhlaqur and Sarwat, and my grandparents, who were a constant source of inspiration for pursuing higher studies. I am dedicating this dissertation to them. From early childhood, they not only instilled the importance of higher education within me but also facilitated every opportunity for higher education within their capability. Especially, my parents have made countless sacrifices to prioritize my (and my siblings') education over anything else. None of my success would be possible without their support.

I am also grateful to many people outside of the university who enriched my life in Canada. Reaz, Shapla, Abrar and Shahriar have been constantly extending their geniality over the years and provided me with a home away from home. Nahian, Arup, Ishtiaque, Nashid, Kashpia, and Sharfan, my housemates and neighbors at different times, made my stay in Waterloo filled with joy. Even after leaving Waterloo, Nashid and Kashipa's little one Sharfan kept holding "bi-weekly meetings", which was a breath of fresh air during the pandemic. The hangouts, adventures, trips, and restaurant hopping around Canada and the world with Swastie, Zulfa, Shahed, Noah, Bushra, Ameer, Sakib, and Hyame are some of the fondest memories that I will be taking from my time during PhD.

My research was supported by several funding sources, including, the Ontario Graduate Scholarship, President's Graduate Scholarship and Go-Bell Scholarship at University of Waterloo, the Graduate Excellence Scholarship at the David R. Chriton School of Computer Science, MITACS Globalink Research Award, Graduate Research Scholarships supported by a number of NSERC funded research programs, and the associate team between INRIA RESIST and the University of Waterloo. These financial aids made it possible to complete my degree in a happy and healthy manner. Many of my conference travels were supported in part by the IEEE Communications Society student travel grant and in part by the research travel support provided through the school, faculty and the university, which enabled me to present my research to the community and enrich my professional network.

Dedication

To my parents Akhlaqur Rahman Chowdhury and Sarwat Parveen,
and my grandfather Mohammad Azizul Haque

Table of Contents

List of Tables

List of Figures

4 Dedicated Protection for Survivable Virtual Network Embedding

5 Adaptive Monitoring of Softwarized Networks

List of Algorithms

Chapter 1

Introduction

1.1 Network Transformation through "Softwarization"

Telecommunications and data center network operators are faced with increasing network management challenges because of: (i) unprecedented growth of network traffic, applications, and users; and (ii) stringent requirements of emerging applications (*e.g.*, VR/AR streaming, tactile Internet, Industry 4.0) in terms of high capacity, ultra-high reliability and low latency [4]. These challenges are often attributed to the fast-paced technological innovation in applications and services significantly impacting the way networking infrastructures are being used. For instance, AT&T, one of the largest telecommunications network operators in the North America, experienced 100,000% increase in network traffic between 2008 and 2016 [5]. Furthermore, major global events such as the COVID-19 pandemic can reshape the way we live, work and interact, resulting in a long-lasting impact on the nature of network traffic [6, 7].

For the years to come, both volume of network traffic and number of connected devices are expected to increase manyfold. For instance, the number of devices connected to IP networks are forecasted to grow from 18.4 billion in 2018 to 29.3 billion in 2023 [8]. Also, with the advent of the fifth-generation (5G) mobile networks, telecommunication operators are expected to support applications with diverse Quality-of-Service (QoS) requirements in terms of bandwidth, latency, reliability, and connection density on the same network [9, 10, 11]. While coping with these growing demands, telecommunications operators have to constantly keep up with the fierce competition from cloud-based over-the-top service providers. To do so, many telecommunication operators and Internet Service Providers (ISPs) are now considering cloud computing as an integral part of their infrastructure. They are transforming their legacy Central Offices (COs), Internet eXchange Points (IXPs) and Points-of-Presences (PoPs) into data

centers [5], in this way realizing the long awaited convergence of Information Technology (IT) and telecommunications [12]. These data centers are leveraged to provide value-added services such as content caching [13] and edge analytics [14], among others. Over-the-top service providers are also forced to redesign both the applications and the infrastructure for keeping up with the new genre of application requirements at scale. Emerging microservices and serverless application architectures, the growing trend for deploying infrastructure closer to the users [15, 16], and large-scale dedicated private networks [17, 18, 19, 20] are mandating new network design and traffic engineering.

Networks are at the center of telecommunications and cloud infrastructures. For the past few decades, the slow innovation in the networking industry has been a major obstacle in scaling and adapting the network infrastructure with evolving user and application needs. This slow pace of innovation is attributed to the closed and physical nature of the networking industry, *i.e.*, network control functions are vertically integrated with packet forwarding hardware with little to no programmability and made available by a handful of vendors. The lack of programmability and open interfaces forced network operators to adapt error-prone manual configuration methods to provision and manage network resources, leading to increased operational complexity and prolonged time to market for new services.

More recently, a major paradigm shift known as *Network Softwarization*, is transforming networks into open, virtualized, programmable, and automated infrastructures. Network softwarization is defined as *"a paradigm where software controls the treatment of flows in the network, adds value to these flows by software processing, and orchestrates the dynamic allocation of resources to meet the needs of customer applications while also promoting energy efficiency through the right-sizing and optimal placement of packet processing in a converged network and cloud infrastructure"* [21]. Networking industry transformation through softwarization can be illustrated by the widespread adoption of the following technological developments.

Software-Defined Networking (SDN) proposes to decouple a network's control plane from the packet forwarding plane (*i.e.*, the data plane), and implements the control plane as a logically centralized software *controller* running on one or more commodity servers [22]. In contrast to traditional networks, where network control is distributed and vertically integrated with the routers and switches, the SDN control plane is separate from the forwarding devices, has a global view of the network, and centrally makes traffic management decisions according to operational policies. The way packet forwarding devices treat network flows is programmed by SDN controllers through well-defined interfaces such as OpenFlow [23]. The capability to program the network enables faster innovation, leading to greater responsiveness to changes, efficiency, and cost effectiveness. Since its inception, SDN's programming capabilities have

grown beyond flow-table programming to protocol independent programmable packet parsing and processing on commodity switches [24].

Network Function Virtualization (NFV) proposes to decouple *Network Functions (NFs)* (*e.g.*, Network Address Translators (NATs), Firewalls, WAN Optimizers) from hardware *middleboxes* [25], and deploy the NFs as Virtual Network Functions (VNFs) on commodity servers [26]. The NFV movement was initiated in response to the high capital and operational expenditure incurred by network operators as a result of the closed and inflexible ecosystem of hardware middleboxes. Despite being an integral part of enterprise and telecommunication networks [27], middleboxes have been vendor specific, proprietary with little to no programmability and vertically integrate packet processing software with the hardware. In addition, they require specially trained personnel for deployment and maintenance. Through the separation of NFs from proprietary hardware middleboxes, NFV promises to reduce capital investment by consolidating multiple NFs on the same commodity hardware, and reduce operational cost by leveraging advances in application orchestration for on-demand service provisioning.

Network Virtualization (NV) is a networking environment that allows coexistence of multiple *Virtual Networks (VNs)*, each tailored to support specific application or service, on a shared physical infrastructure [28]. Since its inception in the mid 2000s to fend off Internet ossification [29], NV has evolved as an enabler of new service offerings for both network and cloud service providers. Recently, NV is gaining more traction because of its importance in 5G network slicing for facilitating the coexistence of applications with diverse requirements such as ultra-high bandwidth, ultra-high reliability and low latency, and massive connectivity [30, 31].

Network softwarization is achieved by the amalgamation of SDN, NFV and NV, enabling on-demand service provisioning and better control over the network resources. Furthermore, softwarization is replacing purpose-built hardware with commodity off-the-shelf hardware with open architecture [5, 32, 33], in this way simplifying network infrastructure and diversifying the supply chain. As a consequence, today network softwarization is experiencing increasing adoption from both large-scale online service providers [17, 18, 19, 20] and telecommunications and Internet service providers [34, 35, 36], and is considered a key enabler for 5G networks [37, 38]. Furthermore, it is creating new revenue streams by enabling service offerings [39, 40, 41] which would have been otherwise very expensive to deploy and hard to manage with traditional networking technologies.

In Figure 1.1, we attempt to present a generalized and simplified view of softwarized networks. The data plane can consists of one or more data centers used for deploying applications

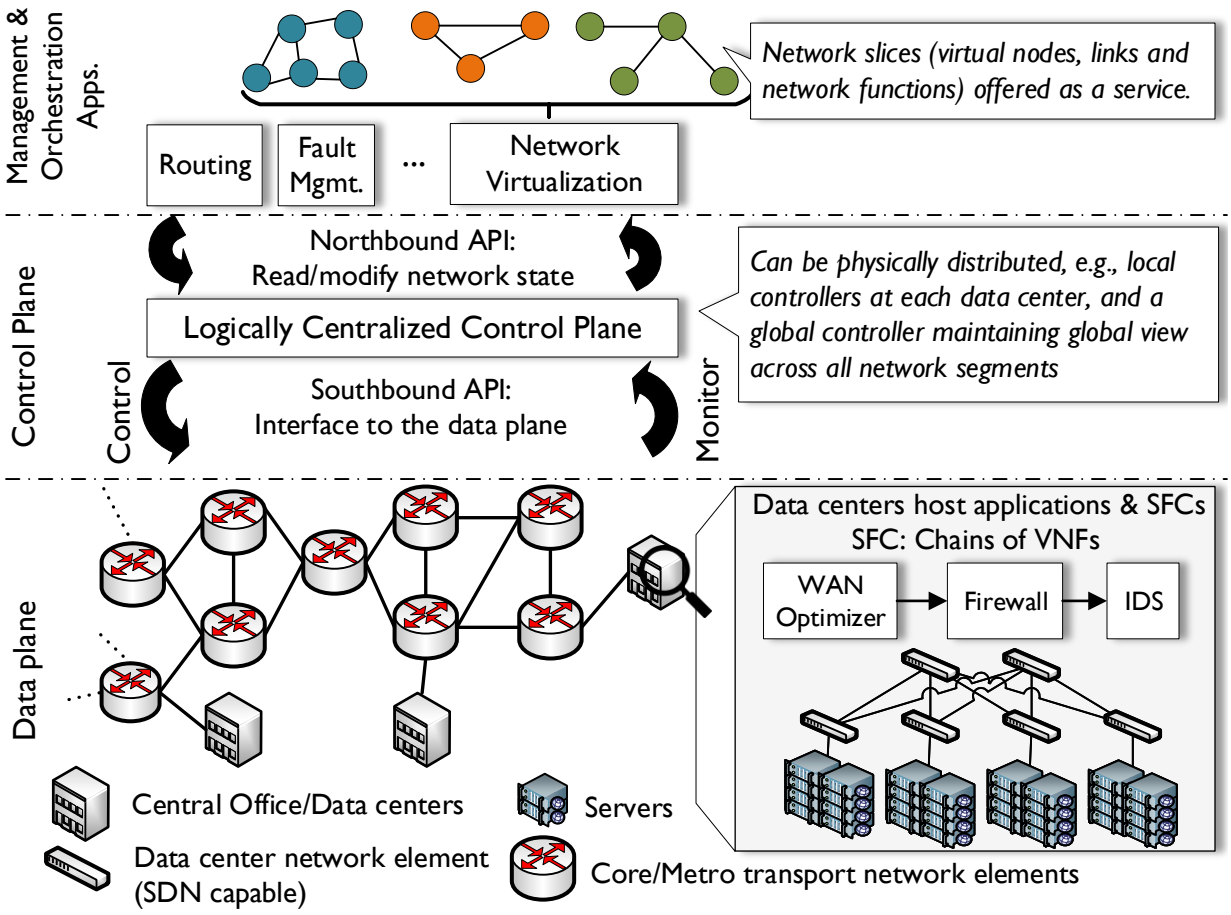


Figure 1.1: A generalized view of softwarized networks

and VNFs providing value added services. Connectivity between these data centers are provided by privately owned or leased core/metro transport networks. Each network segment in the data plane can be composed of off-the-shelf hardware with open interfaces. A logically centralized control plane is responsible for controlling and monitoring the infrastructure through open southbound Application Programming Interfaces (APIs) such as OpenFlow and P4Runtime. The control plane can be physically distributed including a combination of local and global controllers as proposed in the literature [42]. The control plane also provides northbound APIs for developing management and orchestration applications that read and/or modify the network state. NV is one such key application that leverages the logically centralized control plane and its global network view for establishing VNs or network slices and offers them as a service to service providers. Note that such an architecture represents a wide-range of telecommunication and cloud infrastructures. The data centers represent the COs, IXPs, and PoPs owned by the telecommunication operators and ISPs, or the hyper-scale and edge data centers owned by cloud service providers, among others. Depending on the type of network, these data centers can host a wide-range of VNFs such as those from telecommunications and enterprise networks [27].

1.2 Resource Management Challenges

Network softwarization is without doubts reshaping the landscape of and creating new revenue streams for telecommunications and data center networks alike. Softwarization enables network operators to dynamically allocate right-sized resources at the optimal locations for satisfying application and user QoS requirements. However, to effectively leverage the flexibility provided by softwarization, it is of paramount importance to devise the proper mechanisms for allocating resources to different applications and users and for monitoring their usage over time. Sub-optimal resource allocation can result in resource fragmentation and under-utilized infrastructure, which in turn can cause loss of revenue. In the remainder of this section, we discuss some key resource management challenges in SDN, NFV, and NV that need to be addressed in order to realize the full potential of network softwarization.

1.2.1 Fine-grained Resource Allocation and Scaling in NFV

A key advantage of network softwarization is that multiple network functions such as Firewalls, NATs and WAN Optimizers can be consolidated on the same commodity hardware as opposed to relying on purpose-built and closed networking equipment for doing the same. This is especially true for NFV, which promises to reduce the capital and operational expenditure for

network operators by moving packet processing from purpose-built middleboxes to software running on commodity servers (also known as VNFs). However, state-of-the-art NFV platforms (*e.g.*, OPNFV [43], OpenMANO [44], E2 [45]) are *merely replacing monolithic hardware middleboxes with monolithic VNFs*. Clearly, this is a first logical step towards network softwarization. However, common functionality is repeatedly implemented in monolithic VNFs. Repeated execution of such redundant functionality is particularly common when VNFs are chained to realize *Service Function Chains (SFCs)* [46] and results in wasted infrastructure resources.

A fundamental problem with monolithic VNF implementations is that many packet processing tasks such as packet I/O, parsing and classification, and payload inspection are repeated across a wide range of NFs [47, 48, 49]. This has several negative consequences. First, redundant development and optimization effort on these common tasks across different VNFs. Second, monolithic VNFs restrict how many packet processing tasks can be consolidated on the same hardware. For instance, a Firewall and an Intrusion Detection System (IDS), both perform packet classification [48]. Since the VNFs are monolithic, we cannot consolidate packet classification as a single function, allocate just enough resources for processing the cumulative traffic of the Firewall and the IDS, and deploy the classifier as a single entity. Third, monolithic VNFs impose coarse-grained resource allocation and scaling. Finally, when VNFs are chained to form SFCs, executing these redundant functionalities results in unnecessary processing overhead (shown to exceed 25% for some SFCs [47]). This non-exhaustive list of issues stresses the need to rethink how VNFs can be developed and orchestrated for agile service creation and scaling [50]. In this thesis, we pose and address the following research question:

***(Q1)** What is an appropriate software architecture for VNFs that will enable better function consolidation on the same hardware, and finer-grained resource allocation and scaling while maintaining the same level of performance as state-of-the-art approaches?*

1.2.2 Virtual Network Embedding over Transport SDN

Infrastructure providers such as data center network operators, and telecommunications and Internet service providers are rolling out network virtualization technologies to offer slices of their networking infrastructure to service providers [51, 52]. Even the long-haul connectivity providers, *i.e.*, the transport network operators are leveraging SDN to offer full-fledged VNs to their customers [53, 54]. This next generation of transport network, also known as *Transport SDN (T-SDN)*, leverages SDN technology to separate the control plane from the data plane

(typically realized through a combination of packet and optical communications) for flexible management and better automation. T-SDN enables the coexistence of multiple customers with full-fledged VNs in lieu of providing traditional point-to-point connectivity services. The customers can then deploy their own routing and traffic engineering solutions, this way achieving customized control over their own network slice(s).

The benefits from T-SDN virtualization come at the cost of additional resource management challenges for the infrastructure provider. A fundamental and well studied problem in NV is to efficiently embed the virtual nodes and links of a VN request from a service provider on the nodes and paths of the *Substrate Network (SN)*, also known as the *Virtual Network Embedding (VNE)* problem [28]. Typical objectives for VNE include maximizing the number of embedded VNs [55] and minimizing the resource provisioning cost on the SN [56, 57, 58]. Since T-SDN virtualization is an emerging area, several novel and practically important resource allocation problems exist. However, there is a lack of systematic approach to optimally solve these problems in the literature. In the following, we discuss two such challenges for T-SDN virtualization.

The first challenge concerns the choice of SN technology for deploying T-SDN. Several T-SDN deployment choices exist, among which multi-layer IP-over-Optical networks are becoming a popular deployment choice. The VNE research literature has paid significantly lesser attention to multi-layer substrates networks [2] compared to single-layer networks [59], which will be our focus in this thesis. The second challenge concerns guaranteeing VN survivability against failures in T-SDN. Customer VNs typically carry high volumes of traffic at high speed, and usually have Service Level Agreements (SLAs) with the infrastructure provider for recovery from substrate failures within tens of milliseconds [60, 61]. We will discuss the gap in available resource allocation algorithms for satisfying such tight SLAs.

Multi-layer IP-over-Optical Substrate Network

Multi-layer IP-over-Optical networks are becoming a popular choice for deploying transport networks [62]. Such multi-layer networks typically consist of an optical substrate for the physical communication with an IP overlay on top [63]. This network model is being increasingly adopted for transport networks as it offers the best of both worlds, *i.e.*, the flexibility in addressing, resource allocation, and traffic engineering of IP networks along with the high capacity provided by optical networks. Despite their increasing popularity, research on addressing resource provisioning challenges for virtualizing such networks is still in early stages. As mentioned earlier, a classical resource provisioning problem in NV is the VNE problem. VNE has been extensively studied for single-layer SNs [59] with significantly lesser attention paid to the multi-layer network substrates [2].

Solving the VNE problem for multi-layer networks raises many unique challenges due to the topological flexibility offered by such networks [64]. Specifically, the IP network is dynamic, *i.e.*, new IP links can be established when needed by provisioning necessary capacity from the optical network. Such flexibility can be exploited if residual resources in the IP layer are insufficient to admit a new VN, or to reduce the cost of VN embedding by creating new IP links that reduce network diameter. Provisioning new IP links in optical networks has been a tedious and manual task with a long turnaround time. However, with the advances in optical networking technologies [65] and centralized optical control plane in T-SDN [53, 66, 67, 68], such provisioning tasks are becoming more and more automated. Even then, one should not abuse such capability to sporadically establish new IP links since it still remains more expensive than embedding virtual links on existing IP links. In this context, we pose the following research question:

(Q2) *How can we leverage the topological flexibility of multi-layer IP-over-Optical T-SDN and: (i) strike a balance between obtaining a low cost VN embedding while minimizing the establishment of new IP links; (ii) simultaneously decide on the creation of new IP links and their embedding on the optical network.*

Guaranteeing VN Survivability against Transport Network Failures

One particular aspect of VNE is to take the possibility of SN failures into account, known as the *Survivable Virtual Network Embedding (SVNE)* problem [69]. Protection and restoration mechanisms exist in the literature for SVNE. Restoration approaches reactively take action after a failure has occurred, while protection approaches pro-actively provision backup resources when a VN is embedded. As mentioned earlier, transport network customer VNs typically have SLAs with the infrastructure provider for recovery from substrate failures within tens of milliseconds [60, 61]. One way to satisfy such tight SLA is that the infrastructure provider provisions dedicated backup resources for the entire VN topology (*i.e.*, for each virtual node and virtual link in a VN request). Backup of the entire VN topology can be later used for immediate recovery from a substrate failure [70]. Otherwise, a prolonged recovery time can lead to service disruption, leading to revenue and reputation loss for the infrastructure provider.

The protection scheme described above is known as the $1 + 1$-protection scheme. It has its roots back to Wavelength Division Multiplexing (WDM) optical networks where each primary lightpath is established with a dedicated and disjoint backup path for recovering fiber cuts within tens of milliseconds [60, 61]. In case of T-SDN, multiple customers with full-fledged

VNs will coexist instead of traditional end-to-end connectivity as in WDM networks. However, such fast recovery with dedicated backup comes at the expense of provisioning idle backup network resources. The research literature lacks a systematic approach to solving the problem with optimal resource footprint. Furthermore, a scalable heuristic that jointly maps the virtual nodes and links with dedicated protection is also missing from the literature. Finally, relevant literature [3] shows that sequentially embedding the primary and backup can lead to failure in embedding even though a feasible embedding exists, which adds another dimension for designing a resource efficient heuristic. In this context, we pose the following question:

(Q3) *How can we simultaneously compute the primary and the backup embedding of a VN for dedicated protection while jointly determining virtual node and virtual link embedding and incurring the minimum resource footprint in the substrate network?*

1.2.3 Accuracy – Overhead Trade-off in Network Monitoring

Network monitoring is fundamental to network management and is the basis of many network Operations, Administration and Management (OAM) activities such as fault management [71], traffic engineering [72], load balancing [73, 74], threat detection and mitigation [75, 76], and capacity planning, accounting and billing [77], among others. Traditional IP network monitoring solutions such as NetFlow, sFlow and SNMP are hard to implement, and often require deploying proprietary hardware [78, 79, 80]. A fundamental issue in network monitoring is the trade-off between network monitoring overhead and the accuracy of the network view constructed from network monitoring data. Many contemporary network monitoring solutions sacrifice accuracy in exchange of limiting the monitoring overhead. For instance, the aforementioned IP network monitoring solutions heavily sample the packets (*e.g.*, sFlow recommends sampling 1 in every few thousand packets [81]) for coping with the high memory and computational overhead stemming from monitoring every packet in the network [82, 83].

The advent of SDN has addressed many shortcomings of traditional IP network monitoring solutions. OpenFlow, the *de facto* standard for SDN, defines per-flow (a *flow* is identified by a tuple of packet header fields as defined by OpenFlow standard) statistics counters in the data plane that can be configured and read from the logically centralized SDN controller. The SDN controller can program the flow tables in the data plane through the OpenFlow protocol and can specify flow signatures to monitor. These counters can be later read by the controller for constructing a global view of the network. Such flexible flow table program-

ming and monitoring capabilities have resulted in many OpenFlow based monitoring applications such as heavy-hitter detection [84], change detection [84], anomaly detection [85], traffic matrix computation [86], distributed systems monitoring [87], among others. However, the data plane flow tables are typically implemented using high-speed and power hungry Ternary Content-Addressable Memory (TCAM) [88], hence, is usually a limited resources. Consequently, the controller trades off the number of flows to monitor with the memory usage in the data plane [84, 89]. Furthermore, constructing a near real-time view of the network flows in the control plane requires very frequently reading the statistics counters, which comes at the cost of increased control plane bandwidth usage and message processing overhead at the controller.

Network monitoring solutions until the first generation of SDN have been *pull-based*, *i.e.*, the centralized control plane or a network management system periodically reads the counters from the network devices [72, 90]. A drawback of pull-based monitoring is the coarse time granularity of monitoring the network, typically in the order of seconds or minutes. In contrast, the recently emerging push-based *streaming telemetry* is enabling the network devices to directly stream telemetry information to data collection and analytics engines [91, 92, 93], providing near real-time and microscopic visibility into the network. Along the same vein of streaming telemetry, *In-band Network Telemetry (INT)* [94] has been recently proposed as a means to obtain per-packet real time view of the network. INT is an effort to enable network devices (*e.g.*, software and hardware switches, Network Interface Cards (NICs)) to embed device internal state such as packet processing latency and switch queue occupancy into each passing packet, consequently, facilitating a real-time and microscopic view into network traffic. Such fine-grained telemetry capability is enabling new use-cases such as pin-pointing the root cause of congestion and packet drops through switch queue profiling [95] and per-packet fine-grained feedback to support low-latency data center transport [96], which are otherwise difficult to perform with traditional network monitoring. As of today, INT is supported by commodity hardware such as fixed-function and programmable switches [97, 95], and SmartNICs [98, 99], and is being deployed in production telecommunications and data center networks [100, 96].

The microscopic telemetry capabilities enabled by INT come at the expense of increased data plane overhead [101, 102]. This overhead is attributed to each INT capable network device on a packet's path augmenting the packet with telemetry data, thus increasing the packet's size in proportion to the path length. For instance, collecting three telemetry data items on a 5-hop path in a data center results in more than 40% additional bits added to a packet compared to the original packet size (details in Section 5.4.1). Packet size increase for carrying telemetry data can have several negative consequences including, reduction in data plane goodput [102] and negative impact on application latency due to unnecessary packet fragmentation.

The trade-off between accuracy and overhead has been a longstanding issue in network

monitoring. This issue is further aggravated by the flexibility brought by network softwarization. Softwarization is enabling the network operators to program a wide-range of network resources ranging from the software switches, the software network functions, and even the hardware data plane for collecting fine-grained and near real-time network monitoring data. A direct consequence of this advantage is the increased monitoring overhead in both the control and the data planes. In this context, we pose the following question:

(Q4) *How can we construct an accurate and timely view of the network without incurring significant control plane and data plane overhead for network monitoring?*

1.3 Thesis Contributions

This dissertation aims at advancing the state-of-the-art in how resources are allocated and monitored and build the foundation for effective resource management in softwarized networks. We challenge some of the current practices as well as address the shortcomings in how resource allocation is performed and how the network infrastructure is monitored. An overview of the contributions is presented in Figure 1.2. As illustrated in Figure 1.2, this thesis addresses resource management issues encompassing SDN, NFV, and NV, substantially improving the effectiveness of resource allocation and striking a balance between accuracy and overhead in infrastructure monitoring. First, we challenge the current practice of realizing SFCs with monolithic VNFs and propose a microservice-based disaggregated VNF architecture for finer-grained resource allocation and scaling (Q1). Then, we propose optimal solutions and scalable heuristics for the multi-layer VNE (Q2) and 1+1 VN protection problems (Q3) in the context of T-SDN virtualization. Finally, we propose adaptive sampling mechanisms to address the accuracy-overhead trade-off in softwarized network monitoring, focusing on reducing both control and data plane overhead (Q4). We elaborate on the thesis contributions in the following.

1.3.1 A dissaggregated packet processing architecture for NFV

In Chapter 2, we take advantage of the commonality of packet processing tasks among VNFs for addressing the shortcomings of monolithic VNFs. To this end, we propose μNF, a disaggregated packet processing architecture that follows the microservices design principle [103]. We propose to decompose VNFs into independently deployable, loosely-coupled, lightweight,

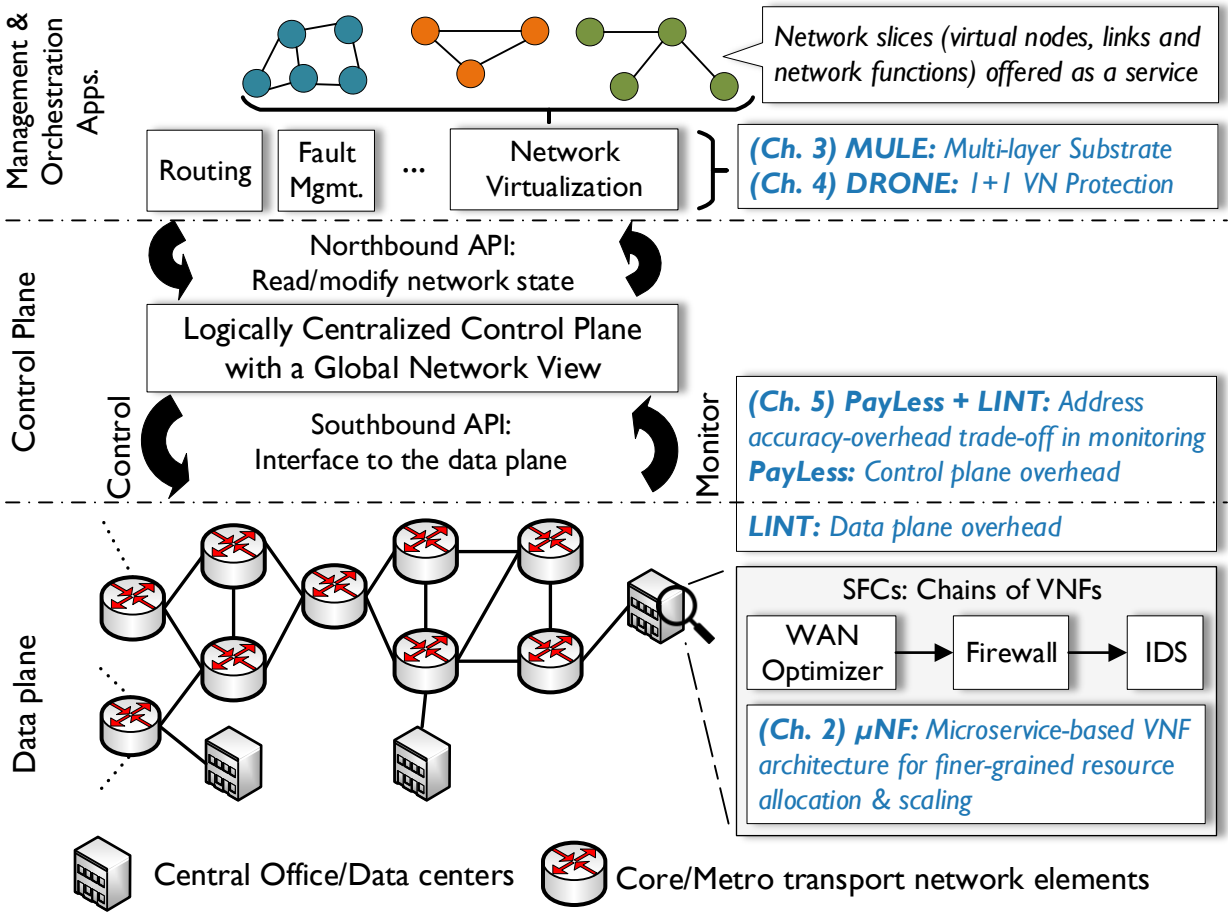


Figure 1.2: Graphical summary of thesis contributions

and reusable packet processors, that we call *MicroNFs* (μNFs for short). VNFs or SFCs are then realized by composing a packet processing pipeline from these independently deployable μNFs. Such decomposition will allow finer-grained resources allocation, independent scaling of μNFs thus increased flexibility, and independent development and maintenance of packet processing components. Specifically, we make the following contributions:

- An architecture for composing VNFs and SFCs from independently deployable, loosely-coupled, lightweight, and reusable components following the microservices design principle [103].
- Implementation of architecture components including the μNFs, communication primi-

tives between μNF, and CPU sharing between μNFs to improve CPU utilization without sacrificing packet processing throughput.

- Optimizations for improving packet processing throughput of μNFs on multi-socket *Non-Uniform Memory Access (NUMA)* machines and latency in μNF-based SFCs.

1.3.2 Multi-layer Virtual Network Embedding

In Chapter 3, we study the ***MU**lti-**L**ayer Virtual Network **E**mbedding (MULE)* problem considering IP-over-Optical SNs. Several deployment choices exist for multi-layer IP-over-Optical networks. Among other options, we focus on multi-layer IP-over-Optical Transport Network (OTN). An OTN is a digital wrapper over circuit switched Dense Wavelength Division Multiplexed (DWDM) optical network with advanced transport capabilities (*e.g.*, traffic grooming without optical-electrical-optical conversion) [104]. We assume the OTN is statically provisioned, *i.e.*, the interfaces on OTN nodes are pre-configured and the corresponding light paths in the DWDM layer are lit in advance to provision fixed bandwidth between OTN nodes, which in turn can be used to provision one or more logical IP links. As a first step towards addressing VNE for multi-layer networks, we limit the scope of this contribution to the case of a *static OTN* and leave the other possible deployment scenarios for future investigation. Specifically, we make the following contributions in this chapter:

- We formulate the optimal solution to MULE (OPT-MULE) as an *Integer Linear Program (ILP)* with the objective of minimizing total resource provisioning cost for embedding the VN while considering the possibility of establishing new IP links when necessary. State-of-the-art in multi-layer VNE [2] does not optimally solve the problem. To the best of our knowledge, this is the first optimal solution to the multi-layer VNE problem for IP-over-OTN networks.

- We propose a heuristic solution to MULE (FAST-MULE) for solving larger problem instances. The heuristic algorithm jointly decides the virtual network embedding, and creation of new IP links and their embedding on the OTN layer by transforming the problem to an instance of maximum network flow problem. We also prove that our heuristic optimally solves the problem for a special class of VNs, namely, star-shaped VNs with uniform bandwidth demand.

1.3.3 Dedicated Protection for Survivable Virtual Network Embedding

In Chapter 4, we study the problem of $1 + 1$-***Pro****tected* ***Vi****rtual* ***N****etwork* ***E****mbedding (1 + 1 − ProViNE)* with the objective of minimizing resource provisioning cost in the SN, while protecting each node and link in a VN request with dedicated backup resources in SN. The primary and backup embeddings need to be disjoint to ensure that a single substrate node failure does not affect both the primary and the backup. If the primary embedding of a VN is affected by a substrate node failure, the service provider operating on the VN should not incur a significant service disruption typical when migrating the whole or part of the VN to the backup. Indeed, during a single substrate node failure, the disjoint primary and backup embeddings enable the infrastructure provider to instantly switch traffic to the backup embedding without requiring any re-embedding decision. This capability of instantly switching traffic to the backup facilitates fast recovery within tens of milliseconds, which is a typical SLA between transport network providers and customers [60, 61].

To this end, we propose ***D****edicated P****ro****tection for Virtual* ***N****etwork* ***E****mbedding (DRONE)*, a suite of solutions for 1 + 1 − ProViNE. DRONE guarantees a VN to survive under a single physical node failure. We focus on single node failure scenario since it is the most probable case [105, 106], and leave the multiple failure scenario for future investigation. Specifically, we make the following contributions:

- We formulate the optimal solution of 1 + 1 − ProViNE (OPT-DRONE) as an ILP, improving on the quadratic formulation from previous work [3]. We also show that optimally solving 1 + 1 − ProViNE is at least as hard as jointly solving balanced graph partitioning and minimum unsplittable flow problems, both of which are NP-Hard [107, 108].
- We propose a heuristic algorithm (FAST-DRONE) for finding solutions to solve larger instances of 1 + 1 − ProViNE in a reasonable time. Our heuristic algorithm iteratively partitions the SN and jointly embeds the nodes and links from the VN request in each partition, in this way, jointly computes both the primary and backup embedding.

1.3.4 Adaptive Monitoring of Softwarized Networks

In Chapter 5, we study the problem of balancing network monitoring overhead and the accuracy of network view constructed from monitoring data in the context of softwarized networks. We employ the general principle of adapting the sampling frequency of collecting monitoring data in such a way that we capture the interesting network events as much as possible without incurring a substantial overhead. Our work in Chapter 5 is divided into two parts. The first

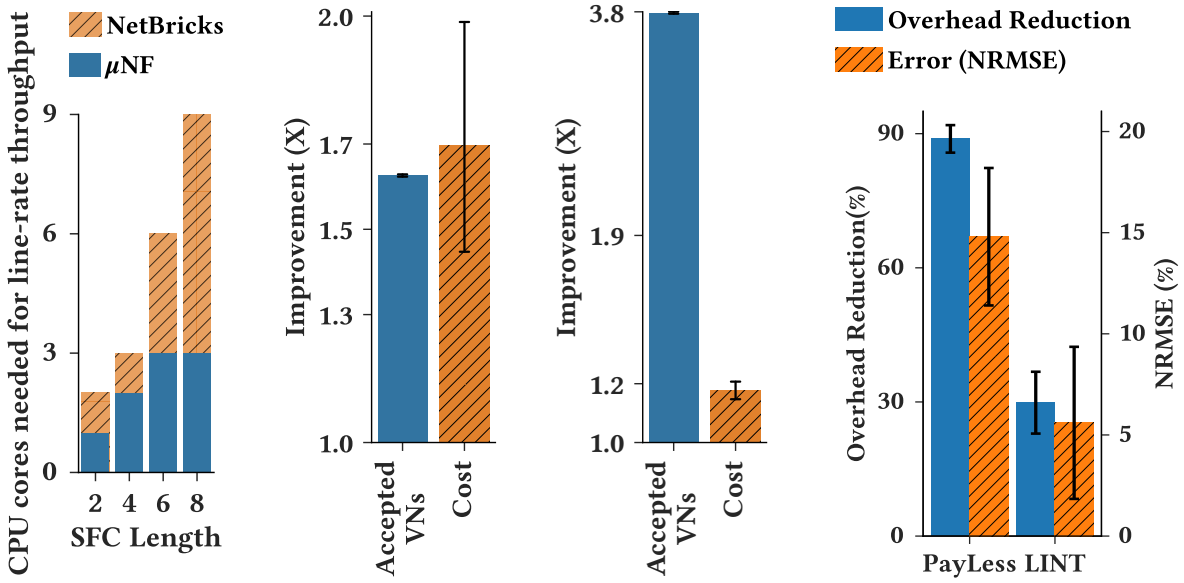
part is concerned with reducing the control plane overhead for SDN monitoring. In this context, we propose PayLess, a traffic intensity-aware variable frequency monitoring algorithm for OpenFlow networks. PayLess adjusts the SDN controller's polling frequency for collecting flow statistics counters from the data plane devices based on how the traffic intensity of the flows change over time. In this way, PayLess is capable of avoiding unnecessarily polling the network while not missing out on interesting network events. The second part of the work is concerned with reducing the data plane overhead of INT enabled by programmable PISA devices and the domain specific P4 data plane programming language. In this context, we present LINT, an accuracy-adaptive and lightweight INT mechanism that runs in the data plane. Specifically, we have the following contributions in Chapter 5:

- We propose PayLess, a traffic intensity-aware variable frequency algorithm for reducing the control plane overhead of OpenFlow network monitoring. We implement PayLess in Floodlight OpenFlow controller and demonstrate its effectiveness through a network link monitoring application on Mininet [109].
- We propose LINT, an accuracy-adaptive and lightweight INT mechanism for programmable data planes. Network devices employing LINT independently decide on selectively reporting telemetry data on passing packets, without any explicit coordination and intervention from a control plane. We evaluate LINT using publicly available network traces and employing a combination of network emulation (using Mininet [109] and the P4 reference software switch bmv2 [110]) and simulation.

1.3.5 Summary of Results

We have evaluated our solutions using a variety of methods including, testbed experiments, network emulation and simulations. We leveraged publicly available real network topologies and real traffic traces when possible. In many cases, we also implemented state-of-the-art solutions and compared our contributions against them. In this section and in Figure 1.3, we summarize and present some key results of our contributions.

We first demonstrate the finer-grained resource allocation capability of μNF compared to state-of-the-art run-to-completion monolithic SFC deployment system NetBricks [1] in Figure 1.3(a). We deploy SFCs of varying lengths using both μNF and NetBricks on a testbed and measure the number of CPU cores they require for sustaining line-rate packet processing throughput (for the smallest packet size). We found μNF to be always using the same or lesser number of CPU cores compared to NetBricks. Furthermore, the CPU core consumption gap between μNF and NetBricks widens as the SFCs become longer.



(a) μNF vs. NetBricks [1]

(b) FAST-MULE vs. D-VNE [2]

(c) FAST-DRONE vs. PAR [3]

(d) Effectiveness of PayLess & LINT

Figure 1.3: Highlights of thesis contributions: (a) effectiveness of μNF in enabling finer-grained resource allocation compared to NetBricks [1]; (b) VN acceptance and embedding cost comparison between FAST-MULE and D-VNE [2]; (c) VN acceptance and embedding cost comparison between FAST-DRONE and PAR [3]; and (d) effectiveness of PayLess and LINT in reducing control and data plane overhead, respectively.

Following the effectiveness of fine-grained resource allocation, we demonstrate the performance of the resource allocation heuristics proposed for MULE and 1 + 1 – ProViNE in Figure 1.3(b) and Figure 1.3(c), respectively. In both cases, we extensively evaluate the solutions using realistic network topologies under different settings and compare the results against state-of-the-art heuristics. In case of MULE, our proposed heuristic, FAST-MULE, accepts $\approx$1.6$\times$ more VNs and incurs $\approx$1.7$\times$ less resource provisioning cost on average compared to state-of-the-art heuristic [2]. FAST-DRONE also outperforms the corresponding state-of-the-art heuristic [3] by accepting $\approx$3.8$\times$ more VNs and incurring $\approx$1.17$\times$ less resource provisioning cost on average while providing dedicated protection to VNs.

Finally, Figure 1.3(d) shows the effectiveness of PayLess and LINT in reducing control and data plane overhead for softwarized network monitoring, respectively. We evaluate our contributions using network emulation and simulation using both synthetic and real traffic traces.

Our key results are: (i) PayLess can reduce control plane messaging overhead by more than 80% and incurs ≈20% normalized root means squared error (NRMSE) of link utilization measurement on average compared to periodically polling the switches in 250 ms intervals; and (ii) on average, LINT reduces the data plane overhead of INT by ≈30% while incurring ≈5% NRMSE for switch processing latency measurement.

1.4 Thesis Organization

We first present a disaggregated VNF architecture to enable fine-grained resource allocation and scaling in Chapter 2. Following the disaggregated VNF architecture, we shift our focus to addressing two novel resource allocation problems arising from the virtualization of transport networks. The first problem, multi-layer virtual network embedding, focusing on efficiently allocating virtual network resources from a softwarized IP-over-OTN transport network is addressed in Chapter 3. Then, in Chapter 4 we present our solutions to the problem of efficiently embedding virtual networks on a T-SDN with dedicated topology protection. We dedicate Chapter 5 for addressing another important aspect of resource management, namely, monitoring. Specifically, we strike a balance between the accuracy and overhead of network monitoring considering both control and data plane overheads. Finally, we conclude with some future research directions in Chapter 6.

Chapter 2

A Disaggregated Virtual Network Function Architecture

2.1 Introduction

In this chapter, we aim at building Virtual Network Functions (VNFs) from simple building blocks by taking advantage of the commonality of packet processing tasks, following the microservices architecture [111, 112]. We propose μNF that takes the disaggregation of middleboxes one step further and decomposes VNFs into independently deployable, loosely-coupled, lightweight, and reusable packet processors, that we call *MicroNFs* (μNFs for short). VNFs or Service Function Chains (SFCs) are then realized by composing a packet processing pipeline from these independently deployable μNFs. Such decomposition will allow finer-grained resources allocation, independent scaling of μNFs thus increased flexibility, and independent development and maintenance of packet processing components.

μNF is built on the thesis of CoMb [47] that consolidating common packet processing tasks from multiple NFs may lead to better resource utilization. However, CoMb did not address the engineering challenges for realizing the data plane of such a system (*e.g.*, software architecture, performance optimizations), which is our key contribution. Specifically, we have the following contributions in this chapter:

- A quantitative study to demonstrate how repeated application of common packet processing tasks in an SFC can affect CPU resource utilization.
- An architecture for composing VNFs and SFCs from independently deployable, loosely-coupled, lightweight, and reusable components that we call μNFs.

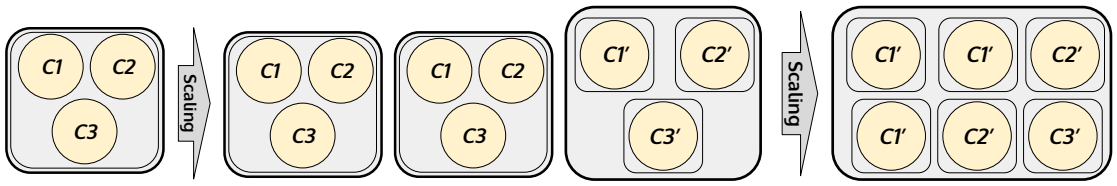
- Implementation of architecture components including the μNFs, communication primitives between μNF, and CPU sharing between μNFs to improve CPU utilization without sacrificing packet processing throughput.
- Optimizations for improving packet processing throughput of μNFs on multi-socket Non-uniform Memory Access (NUMA) machines, and packet processing latency in μNF-based network services.
- Evaluation of the system through testbed experiments. Our key findings are: (i) compared to an SFC composed from monolithic VNFs, μNFs can achieve the same throughput using less CPU cycles per packet on average; (ii) μNFs can sustain the same packet processing throughput as the state-of-the-art run-to-completion VNF architecture [1], while using lesser number of CPU cores.

The rest of this chapter is organized as follows. First, we provide a brief overview of microservices software architecture that inspired the design of μNF in Section 2.2. Then, we motivate the need for VNF disaggregation through both qualitative and quantitative study in Section 2.3. Then we describe our design goals and choices in Section 2.4 followed by a detailed description of the disaggregated VNF architecture in Section 2.5. We present the implementation detail of different system components in Section 2.7. Then we discuss the optimizations that we performed for improving packet processing throughput on multi-socket NUMA machines and packet processing latency in Section 2.6. We present the results from testbed evaluation in Section 2.8. Finally, we summarize the chapter contributions in Section 2.10.

2.2 The Microservices Software Architecture

The National Institute of Standards and Technology (NIST) defines microservices as follows. *"A microservice is a basic element that results from the architectural decomposition of an application's components into loosely coupled patterns consisting of self-contained services that communicate with each other using a standard communications protocol and a set of well-defined APIs, independent of any vendor, product or technology"* [111]. From this definition, we draw the following key features of microservices:

- **Self-contained:** A microservice typically performs a single task and does not depend on other microservices for performing its task.
- **Loosely coupled:** Generally, a microservice does not require tight synchronization with other microservices to operate.



(a) A monolithic application with tightly coupled components, inefficient scaling and coarse grain resource allocation

(b) A microservice-based application with independently scalable microservices, facilitating fine grain resource allocation

Figure 2.1: Monolithic vs. microservice-based application

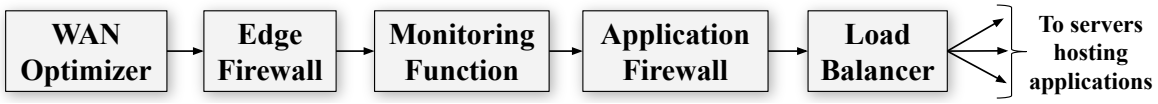
- **Well defined communication interfaces:** In a microservice architecture, an application completes a series of tasks by different microservices. For this purpose, microservices must exhibit well-defined APIs that an orchestrator and other microservices can use for communication. A popular choice for such interface is the RESTful API. However, message queues and protocol buffers are competing alternatives.

A major advantage of using microservices is the ability to allocate resources at a finer granularity. For example, consider a monolithic application in Figure 2.1(a), composed of tightly coupled components, $C1$, $C2$ and $C3$. A surge in demand requires increased processing in components $C1$ and $C2$ of the application. In this monolithic application, all components need to be scaled because of the tight coupling, resulting in idle resources allocated to $C3$. In contrast, consider Figure 2.1(b), where the same application is re-architected using microservices $C1'$, $C2'$ and $C3'$. Now the scaling of the application is more efficient in terms of resource allocation. Additionally, the self-contained and loosely coupled nature of microservices facilitate independent development and maintenance.

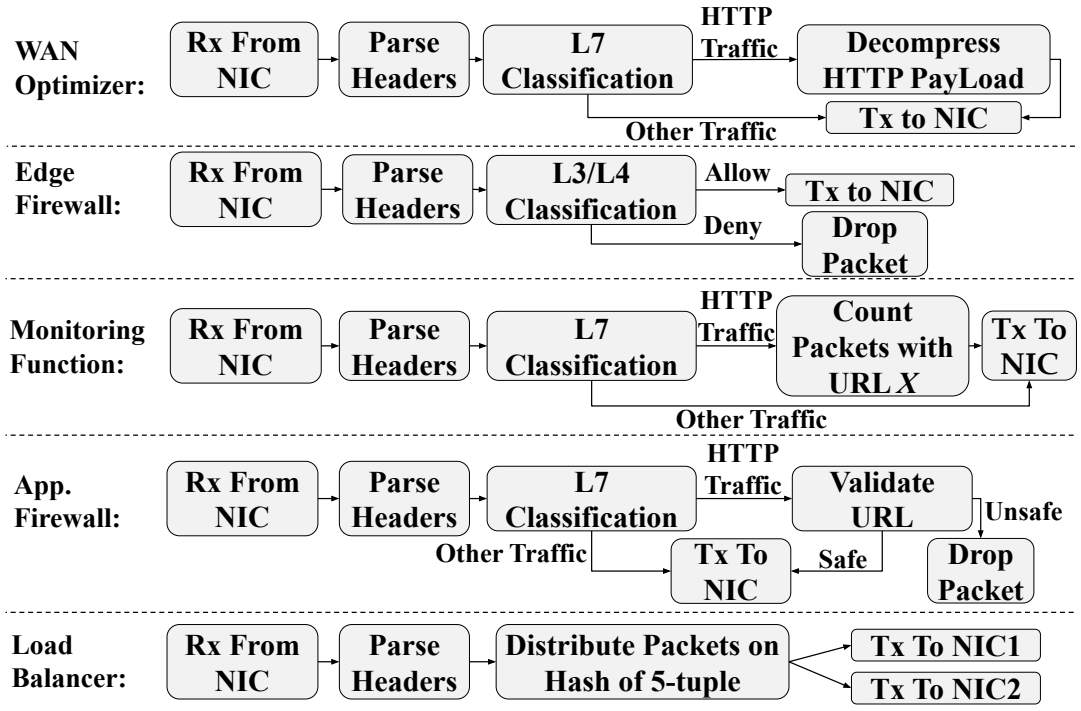
2.3 Motivation

2.3.1 Commonality in Packet Processing Tasks

Our motivation for developing a disaggregated packet processing architecture stems from the observation that many packet processing tasks, such as packet I/O, parsing and classification, and payload inspection are repeated when VNFs are chained in an SFC. We demonstrate this using the SFC in Figure 2.2(a), typically found in enterprise Data Centers (DCs) [113]. This SFC consists of the following VNFs:



(a) Example SFC use case from [113]



(b) Functional decomposition of NFs

Figure 2.2: Common packet processing tasks across NFs

- **WAN Optimizer:** Placed at a DC and WAN boundary for optimizing WAN link usage, *e.g.*, compresses/decompresses HTTP payload (*e.g.*, text, image) to reduce WAN traffic [114].
- **Edge Firewall:** Allows or denies packets based on layer 2-4 header signature.
- **Monitoring Function:** Consists of different counters such as a packet size distribution counter, a counter for packets containing certain URLs, *etc.*
- **Application Firewall:** Filters packets based on layer 7 information, *e.g.*, block incoming HTTP requests with embedded SQL injection attacks (similar to [115]).

- **Load Balancer:** Distributes packets to back-end servers based on flow signature.

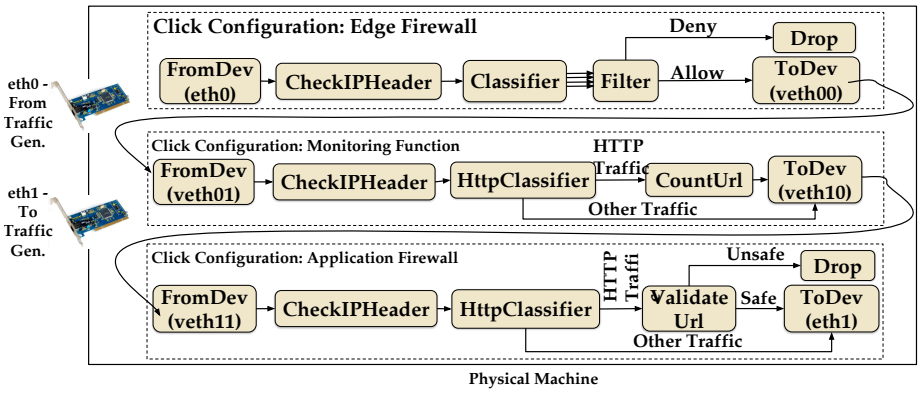
We can decompose these VNFs into smaller packet processing tasks as shown in Figure 2.2(b). Each block in Figure 2.2(b) represents an operation performed on a packet while the arrows represent the flow of operations. Clearly, tasks such as packet I/O, parsing and classifying HTTP packets are repeated in these VNFs. The tight coupling between these packet processing tasks in a monolithic implementation can have the following negative consequences:

- **Overlapped Functionality:** Many of the functionality are common across VNFs. In some cases, the functionality is the same but the parameters are different (*e.g.*, writing a packet but to different Network Interface Cards (NICs)). While in other cases both the functionality and the configuration are exactly the same (*e.g.*, HTTP packet classification performed by both WAN Optimizer and Application Firewall).
- **Wasted CPU Cycles:** Since a functionality is embedded within a monolithic VNF, its execution result is not easily reusable across VNFs (*e.g.*, classification of a packet as HTTP by the WAN Optimizer is not usable by the Application Firewall). Therefore, CPU cycles are wasted due to the repeated execution of the same functionality when a packet goes through the VNFs in an SFC. This is also experimentally validated by prior research, reporting more than 25% CPU cycles are wasted for such redundant processing for certain SFCs [47].
- **Inflexible Scaling:** It is difficult to scale resources for individual functionality since they are embedded into monolithic VNFs. If a functional block in Figure 2.2(b) becomes a bottleneck, the whole VNF needs to be scaled up/out (*e.g.*, there is no easy way to allocate additional CPU resources for performing URL validation in an Application Firewall in Figure 2.2(b) without scaling up/out the whole VNF instance). This requires more resources compared to scaling up/out resources for a single functionality.

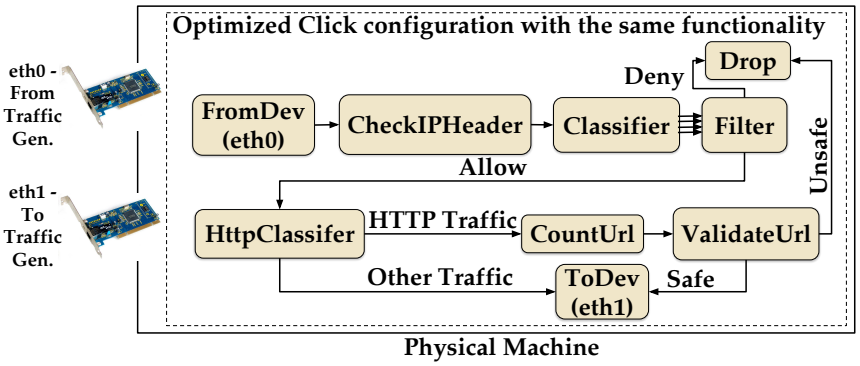
2.3.2 Performance Implications of Monolithic VNFs

We perform an experimental study to demonstrate possible performance implications of repeating common packet processing tasks in an SFC by comparing between the following two deployment configurations: (i) Click [116] based monolithic VNFs chained using virtual Ethernet (veth) pairs (Figure 2.3(a)); and (ii) a single Click configuration implementing the functionality of the same SFC from configuration-i, while removing the repeated common elements (Figure 2.3(b)). For both cases we play the same traffic (HTTP packet trace generated from access log for a moderate size public web-service (≈15K hits/month)) and measure the average CPU

cycles/packet required by each type of Click element. Our objective is to measure the wasted CPU cycles for repeating common tasks across an SFC. Note that this study complements that of the one presented in [47] by demonstrating the impact on an SFC rather than considering single middlebox applications.



(a) Monolithic VNFs chained with veth pairs (configuration-(i))



(b) One single optimized click configuration (configuration-(ii))

Figure 2.3: Motivational experiment scenarios

We deployed the following simplified form of the SFC from Figure 2.2(a): Edge Firewall → Monitoring Function → Application Firewall. We implemented our own Click elements (*HttpClassifier*, *CountUrl*, and *ValidateUrl*) when Click's element library did not have any elements with similar functionality. We also instrumented the Click elements to measure the number of CPU cycles spent in processing each packet.

We present the savings in CPU cycles obtained from removing repeated elements in the optimized configuration, *i.e.*, configuration-(ii) in Table 2.1. We observed a per element savings

Table 2.1: Results from motivational experiment

Click Element Type	CPU Cycles Saved in configuration-(ii)	Element Weight in configuration-(i)
FromDevice	71.9%	0.22%
ToDevice	67.1%	0.25%
CheckIPHeader	65.1%	0.44%
HttpClassifier	48.28%	47.8%
Overall	**29.5%**	–

of up to ≈70%. However, as shown in Table 2.1, not all elements contribute equally to packet processing, hence, the overall gain at the end is 29.5%, which is still significant.

This result further motivates re-architecting VNFs by exploiting the commonality in packet processing in a way to achieve better resource utilization. To this end, we argue in favor of adopting a microservice-like architecture [112] for building VNFs and SFCs. We propose to disaggregate VNFs into independently deployable packet processors, that we call μNFs. VNFs or SFCs can then be realized by orchestrating a packet processing pipeline composed from the μNFs. With this, one can think of applying optimizations such as consolidating multiple instances of a common packet processing function into a single instance for better CPU utilization. We will experimentally demonstrate CPU utilization gains from using a μNF-based SFC over that composed from monolithic VNFs (*i.e.*, configuration-(i)) in Section 2.8.3.

μNFs are to some extent analogous to the VNF Components (VNFCs) defined in the ETSI VNF architectural description [117]. According to [117], a VNF can consist of one or more VNFCs, where each VNFC has a one-to-one correspondence with a virtualization container (*e.g.*, Virtual Machine (VM), OS container). However, a major difference between a μNF and a VNFC is that μNFs are loosely coupled components that can be used across VNF boundaries, whereas VNFCs are not. This property also makes SFC orchestration different in the case of μNFs since there are more optimization opportunities for μNFs compared to VNFCs. We illustrate some of these optimization opportunities through an example in Figure 2.4. This figure represents a possible optimized realization of the SFC from Figure 2.2(a) using μNFs. Assuming that mechanisms to propagate output of one μNF to the others exist, we can consolidate packet processing tasks such as packet I/O, header parsing and HTTP packet classification. Furthermore, some μNFs can be executed in parallel, such as the tasks performed by the monitoring functions that do not modify packets. These optimizations are only possible due to the inherent loosely coupled characteristic of μNFs.

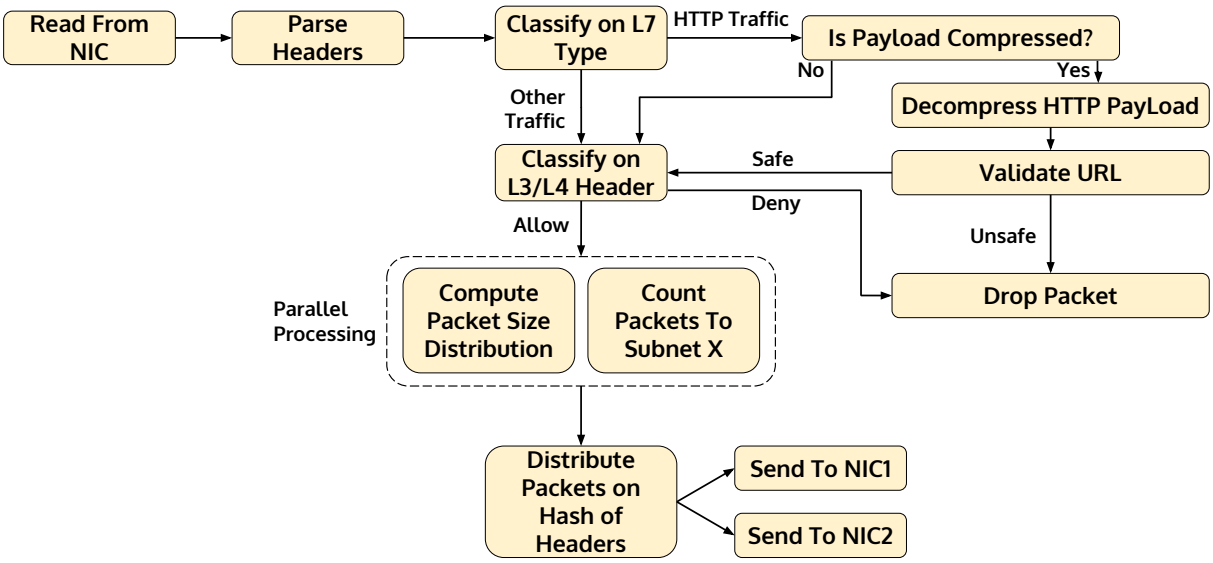


Figure 2.4: Microservice-based realization of the SFC from Figure 2.2(a)

2.4 Design Goals and Choices

Our objective is to re-architect the VNFs by exploiting their overlapping functionality enabling finer-grained resource allocation and achieving better resource utilization. To achieve these objectives we start with the following design goals:

- **Reusability** Frequently appearing packet processing functions should be developed once and shared across VNFs.
- **Loose-coupling**: Packet processing functions should not be tightly coupled, so that they can be deployed and scaled independently, allowing fine-grained resource allocation.
- **Transparency**: Implementation of a packet processing function should not be affected by their communication pattern (*e.g.*, one-to-one, one-to-many, *etc.*).
- **Lightweight communication primitives**: Communication between packet processing elements should not incur significant overhead hurting the overall performance.

The first goal can be achieved by dividing large packet processing software into smaller packet processing tasks or functionality. Then to achieve the rest of the goals we have the following two design alternatives [118]:

Run-to-completion: Packet processors are implemented as a set of identical threads or processes, each implementing the entire packet processing logic (*i.e.*, an NF or even an SFC).

Pipelining: Packet processors are implemented by composing a pipeline of heterogeneous threads or processes, each performing a specific packet processing task.

The state-of-the-art modular VNF designs such as ClickOS [119] and NetBricks [1] have adopted a run to completion model, where packets are passed between different functions in the same address space and processed in a single thread or process. When more processing capacity is required, the whole VNF (or SFC) instance is scaled out and traffic is split between the instances using NIC features such as Receive Side Scaling (RSS) [120]. One limitation of this model is that it is hard to right size resource allocation to individual components because of the tight coupling between them. In contrast, pipelining mode satisfies more of our design goals. Individual components can be allocated their own resource, independently deployed and scaled (loose-coupling), and it is easier to decouple how elements process packets from their underlying communication pattern (transparency).

2.5 System Description

2.5.1 Assumptions

We assume that the network operator owning the infrastructure has control over the VNFs that are being deployed. These VNFs can be deployed at the operator central offices converted into edge data centers [5]. When SFCs are deployed inside these edge data centers their VNFs are typically in the same layer 2 domain.

We do not consider Virtual Machines (VMs) as the choice of deployment for individual μNFs since that would add a significant overhead for μNF to μNF communication [1]. Moreover, we also do not require separate OSs and kernel features for deploying the μNFs, which is typically provided by VMs. Rather we choose using either processes or containers for μNF deployment. At this point we leave the choice of using processes or containers to the network operator since our evaluation results demonstrated similar performance.

We assume that the μNF descriptions (*e.g.*, what type of operation the μNF performs on what part of the packet header or payload) and template for composing VNFs from μNFs will be provided by the VNF providers. The SFC request will come from the network operator. Currently, we use JSON format for SFC specification. However, we do not restrict ourselves as to what can be used for specifying SFCs. We plan to support standards such as TOSCA [121] and YANG [122] in future.

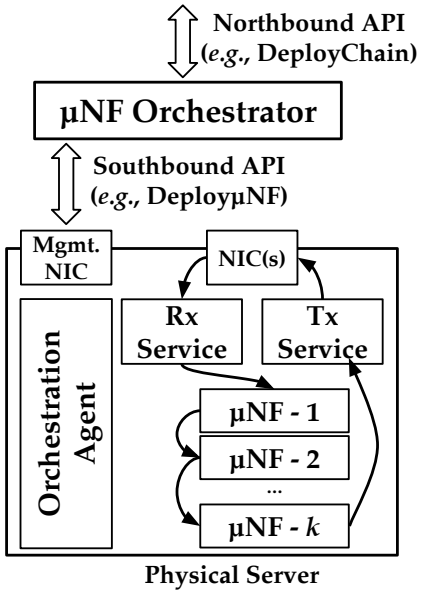


Figure 2.5: System components

Finally, we assume that the μNF developers will provide configuration generator for each μNF. This will generate the necessary configuration for a μNF (*e.g.*, the types of communication primitives), when presented with a μNF type and its connectivity with neighboring μNFs.

2.5.2 System Architecture: Birds Eye View

A high level view of our system is presented in Figure 2.5. It comprises the following components: a *μNF orchestrator*, per physical server *orchestration agent*, *μNFs*, and Rx and Tx services for reading packets from and to the NICs, respectively. The northbound API facilitates SFC lifecycle management and monitoring, and allows network operators to interact with the system. The μNF orchestrator is responsible for making global decisions such as μNF placement across physical servers to realize SFCs and make μNF migration decisions, among others.

The orchestration agent acts as the local orchestration endpoint for a given machine. A southbound API between the global orchestrator and orchestration agents facilitates their communication. For example, the μNF orchestrator can use the southbound API for requesting local orchestration agents to allocate resources for μNFs, deploying μNFs with proper configuration and create the communication primitives for μNF to μNF communication.

The smallest deployable units in the system are the μNFs. μNFs usually perform a specific

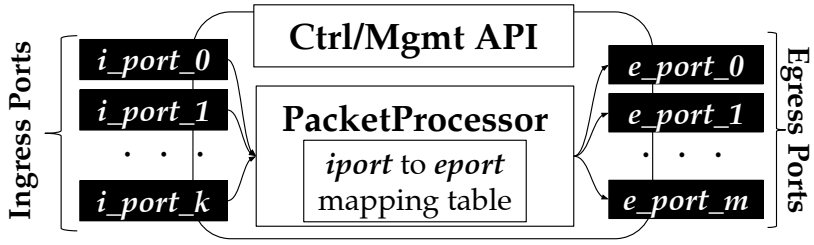


Figure 2.6: μNF architecture

packet processing task and are independently deployable loosely-coupled entities. As described earlier in Section 2.4, one of our design goals is to keep the μNFs simple and keep the communication pattern between μNFs transparent from how they process the packets.

Finally, we have two special μNFs, namely the Rx and Tx services, responsible for reading packets from and writing packets to the NIC, respectively. These two services collectively form a lightweight software data path for the μNFs. By isolating these two services from the μNFs we have the flexibility to adjust I/O batch sizes according to the consumption/production rate of the μNFs. Moreover, such separation allows us to make the operations on hardware transparent to other packet processing μNFs.

2.5.3 System Components

μNF Orchestrator: The μNF orchestrator is responsible for realizing an SFC by orchestrating a packet processing pipeline consisting of μNFs across multiple machines. Network operators can interact with the orchestrator through a north-bound API. The orchestrator is also responsible for global management decisions such as handling machine failures and making scaling decision.

μNF Orchestration Agent: μNF orchestration agent is the local orchestration endpoint on a physical machine. It has a northbound API for the μNF orchestrator to act on it. The agent is responsible for performing local actions such as deploying μNFs, creating communication primitives to enable inter μNF communication on the same machine, *etc.*

μNFs: A μNF is the unit of packet processing in the system as well as the unit of deployment and resource allocation. It consists of a number of *IngressPort*s, a number of *EgressPort* and a *PacketProcessor* (Figure 2.6). The IngressPorts and EgressPorts provide methods to pull packets

from and push packets to the previous and the next μNF in the packet processing pipeline, respectively. When μNFs from different VNFs are consolidated, the IngressPort to EgressPort mapping table helps in routing packets to different branches of the pipeline.

The aforementioned ports are of abstract type and can have different implementations. One of our design goals is to keep packet processing logic of μNFs oblivious to μNF to μNF communication pattern. The port abstraction simplifies μNFs' design and implementation, and keep them loosely coupled with each other. For instance, we implement a *LoadBalancedEgressPort* that has the same interfaces as EgressPort. However, the implementation distributes packets to multiple next-stage IngressPorts in a round-robin fashion. From the μNF's point-of-view this distribution of packets to multiple next stage μNFs remains completely transparent. In Section 2.7 we describe the implementation of different ports in more detail.

Rx Service: Rx service is the interface between host NIC(s) and the μNFs. Rx service keeps hardware specific configurations (*e.g.*, number of NICs, number of Rx queues) and operations (*e.g.*, flow classification in either hardware or software based on NIC capabilities) transparent to the μNFs. The Rx service can be thought of as a lightweight data path (similar to [123], except that complex data path functions are implemented as independent μNFs in our system).

Tx Service: Tx service sits between the μNFs and the host NIC. Common Tx specific tasks such as tagging packets of the same SFC, rewriting destination MAC address with next hop MAC address and writing packets to different NIC queues, are consolidated inside the Tx service.

2.5.4 SFC Deployment

The μNF orchestrator is the entry point for the network operators to deploy an SFC composed of μNFs. One of our goals is to ensure that from the network operators point-of-view the SFC request does not look different from what they are used to seeing, *i.e.*, they should not be required to specify μNF specific configurations. It is up to the orchestrator to determine the optimal composition of μNFs that offers the semantics of the user requested SFC.

Inputs: In what follows, we describe the inputs to the orchestrator in a bottom up fashion:

- **μNF Descriptor:** A μNF descriptor defines different attributes of a μNF. Currently, we support the following attributes: statefulness of the μNF and types of action (*e.g.*, No Operation (NOP), ReadOnly, or ReadWrite) a μNF performs on the packet headers at different protocol layers. For instance, the following is a descriptor for a layer 3-4 classifier:

```
PacketProcessorClass: "TCPIPClassifier"
Stateful: "Yes"
L2Header: "NOP"
L3Header: "ReadOnly"
```

These meta-data about the μNFs assist in performing optimizations (detail discussion in Section 2.6) when composing SFCs from μNFs.

- **VNF templates:** A VNF template is a blueprint of realizing a VNF from μNFs and we represented it by a packet processing graph composed of the constituent μNFs. VNF templates can be considered analogous to VNF descriptors defined in ETSI NFV Management and Orchestration (MANO) specification [124]. A VNF template consists of the nodes of the processing graph (*i.e.*, the μNFs) and the links representing the order of packet processing between μNFs. The links can be labeled with the output of the source μNF for that link. Labels act as a filter, *i.e.*, only packets producing results equal to the label are forwarded along that link. Examples of VNFs and VNF templates are presented in Figure 2.2(b). If we take the Application Firewall VNF from Figure 2.2(b) as an example, it is composed from six independently deployable μNFs. Annotations on the edges represent classification results at different stages, *e.g.*, if a packet contains HTTP payload or not.

- **SFC:** An SFC request is a directed graph, where the nodes are the VNFs and a directed link between two nodes represents the order that traffic should follow. Links can have labels in an SFC indicating VNF specific output. μNF descriptors provided by VNF providers may include more or less information than what we have described. The lesser information they contain, the lesser constraints we may have in placing the constituent μNFs.

Sequence of Operations for SFC Deployment: The μNF orchestrator combines the constituent VNF templates of an SFC, removes redundant μNFs and builds a *μNF forwarding graph* with the same semantics as the SFC request. The graph construction phase can take μNF specific meta-data into account to perform optimizations such as consolidating multiple μNF instances of the same type into one and performing optimization such as parallelizing the executing of multiple μNFs on the same packet whenever possible.

After the μNF orchestrator builds an optimized μNF processing graph and determines the placement of μNFs, it then requests agents on the selected machines to deploy their parts of the graph. μNF orchestrator also generates configuration of each μNF in the graph by leveraging the developer provided configuration generators and provides the agents with these generated

configurations. Upon receiving the μNF processing subgraph and the configurations, the agent first allocates the necessary resources, creates the communication primitives, and deploys and connects the μNFs using the instantiated communication primitives.

2.5.5 Auto-scaling

We propose to use a simple packet drop monitoring based mechanism to take auto-scaling decisions. Once μNFs are deployed, the local agents continuously monitor for packet drops on all EgressPort – IngressPort pairs. A consistent drop indicates that the μNF attached to the IngressPort is not able to match the processing rate of the μNF attached to the EgressPort. This triggers an auto-scaling event in the agent. The agent then spawns another instance of the bottleneck μNF and modifies the corresponding EgressPort in the pair to a LoadBalancedEgressPort (described in Section 2.7.4), which load balances traffic across the scaled-out instances.

However, there is a delay between detecting consistent packet drop and actually deploying another μNF instance to mitigate packet drops. Since the agent is continuously monitoring, it will keep seeing a packet drop during this period and trigger another scale-out event even before the first one completes. To avoid this, we assign a cool down timer to the μNF that is being scaled-out and do not trigger another scale-out event until the timer has expired.

2.6 Optimizations

2.6.1 Pipelined Cache Pre-fetching

One potential issue that might arise from our design of μNF is when using multiple processors in a NUMA configuration. In such configuration, each processor socket has its local memory bank and the access time to local and remote memory banks are not uniform. Processing packets on a NUMA zone (*i.e.*, socket) other than the one where the NIC is attached to has performance implications due to remote memory invocation.

To circumvent the aforementioned issue, we perform a pipelined cache pre-fetching inside every μNF as follows. Before processing a batch of packets, a μNF first pre-fetches a cache-line from the first k packets in the batch. Then it proceeds to process the batch. While packet i from the batch is being processed, a cache-line from packet $i + k$ is pre-fetched into the cache. In this way, when a packet is being processed, the first level cache is very likely to be warm with a cache-line worth data from that packet (which contains the header fields). Thus potentially

increasing the first level cache hit rate and masking the remote memory access latencies to some extent. We experimentally evaluate the impact of this optimization in Section 2.8.2.

2.6.2 Parallel execution of μNFs

In a pipelined packet processing model, the packet processing elements typically operate on a batch of packets in a sequential manner. This is often unavoidable since one μNF only processes the set of packets as determined by the previous stage μNF. For instance, in Figure 2.2(b), the layer 7 classifier in the Application Firewall determines the set of packets to be processed by the URL Validator. However, there are scenarios where sequential processing can be avoided. For example, in the monitoring function from Figure 2.2(b), the counting function performs a read-only operation on the packets. Therefore, if another counting function was part of the Monitoring function, these two could be safely executed in parallel on the same set of packets.

We parallelize the execution of consecutive μNFs from the μNF processing graph that are placed on the same machine by employing techniques similar to the ones discussed in [125, 126, 127]. Parallelization is performed based on the type of operation they perform on the packet header (specified in μNF descriptor). When consecutive μNFs perform read-only operations on the packet header, or operate on disjoint regions of the header, or do not modify the packet stream (*e.g.*, not dropping packets), only then we parallelize their execution and assign them distinct CPU cores on the same NUMA zone.

One issue with parallel execution is to ensure synchronization after the parallel processing stage, *i.e.*, a μNF β that is just after the parallel processing state, should be able to start processing a packet only if the packet has been processed by all the μNFs in the parallel processing stage. Such synchrony is achieved through special IngressPort and EgressPort implementations (details in Section 2.7.4). These ports embed a counter as packet meta-data before parallel execution begins. At the parallel execution stage, each μNF atomically increases the counter after its processing is complete. At μNF β, the IngressPort ensures that only packets with appropriate counter value are passed on to β's PacketProcessor. Moving the synchrony mechanism into ports thus keeps the μNF design simple.

2.7 Implementation

One option for implementing the proposed system is to adapt existing modular packet processing frameworks such as Click [116] to a multi-process model. However, Click comes with a lot

of legacy code, some of which is not useful for our case (*e.g.*, scheduling multiple elements inside a Click binary). Also, Click was originally designed and optimized for a run-to-completion packet processing model, which is fundamentally different from the pipeline model adopted by μNF. Therefore, re-engineering Click and similar systems require significant refactoring of many of their subsystems such as component scheduling and packet transfer to make them efficiently work in a pipeline model. Finally, we wanted to build the system in a way such that it can process packets at 10 Gbps line-rate at least (current *de facto* capacity for commodity NICs) while maximizing CPU usage on the servers. It was becoming cumbersome to optimize Click's performance and refactor its subsystems for pipeline model, hence, we decided to build the system from scratch.

We have implemented a prototype of our system using C++ (agent and μNFs) and Python (orchestrator). Since our focus is more on developing the μNFs and their communication primitives, therefore, our current orchestrator is limited in functionality and acts more as a convenience mechanism for testing. We use Intel Data Path Development Kit (DPDK) [128] for kernel bypass packet I/O and `hugetlbfs` [129] for sharing memory between μNFs. Before describing the implementation details of our system, we first give a brief overview of DPDK.

Intel DPDK Overview

Intel DPDK [128] is a set of libraries for facilitating fast packet processing in the user-space. DPDK contains libraries for kernel-bypass packet I/O, lockless multi-producer multi-consumer circular queues (DPDK `rte_ring` library), and memory management (DPDK `rte_mempool` library), among others. The ring library can be used to create shared memory based abstractions between packet processors for zero-copy packet exchange. DPDK also ships with a set of NIC specific poll-mode drivers (PMDs) for packet I/O to/from the NIC.

Packet processing elements built using DPDK run in the user-space, bypasses the kernel and continuously poll the NIC for incoming packets. Poll-mode I/O in DPDK is a departure from the traditional interrupt driven I/O, where I/O operations engage the CPU only when packets become available at the NIC. Upon receiving an I/O interrupt, the CPU switches context from that of the currently running process to that of the interrupt handler in the kernel, performs packet I/O and copies the packets from the kernel-space to the user-space. In contrast, poll-mode I/O always engages a CPU and performs zero copy I/O from the user-space whenever packets become available at the NIC. In this way poll-mode I/O eliminates the need for context switching, executing interrupt handlers, and copying packets from the kernel- to the user-space. By eliminating these overhead among others introduced by interrupts [130, 131], polling incurs very low CPU overhead and low latency, significantly increasing packet processing throughput compared to interrupt driven I/O [132].

In the remainder of this section we describe the implementation of the system components.

2.7.1 Agent

Agents are implemented in C++ and run as primary DPDK processes. During initialization, an agent pre-allocates memory buffers for the NIC to store incoming packets, and exposes an RPC-based control API for the orchestrator. The orchestrator can use this API to deploy part of a μNF processing graph on a machine. When such a request is received by an agent, it deploys the μNFs according to the orchestrator specified configuration and creates the necessary communication primitives (details in Section 2.7.4). Agents also monitor the μNFs and take scaling out decisions.

2.7.2 μNF

μNFs are implemented by leveraging DPDK APIs. Each μNF runs as a stand-alone secondary DPDK process. Since DPDK allows only one process to be the primary, *i.e.*, have the privileges of memory allocation, μNFs run as secondary DPDK processes. When required, μNFs obtain pre-allocated objects from a memory pool shared with the agent. Memory sharing between μNFs and between a μNF and the agent is enabled by hugetlbfs. The hugetlbfs is mounted on a directory accessible to both the μNFs and the agent, and contains virtual to physical memory mapping of the shared memory regions.

One caveat in this shared memory model is that each process should have exactly the same virtual address space layout in order to successfully translate the shared virtual memory to their physical locations. To do so we had to disable Address Space Layout Randomization (ASLR), a Linux kernel feature for preventing buffer overflow attacks [133]. This is a security vulnerability and is a limitation in our current implementation. However, this is also a limitation of the technology at hand and solving it can be an interesting future work.

2.7.3 Rx and Tx Services

In our design, packet I/O is handled by Rx and Tx services in order to hide hardware specifics from the other μNFs. In our prototype implementation, the Rx service runs as a separate thread inside the agent and is pinned to a physical CPU core on the same socket where the NIC's PCIe bus is attached. It receives packets from a NIC queue in batches and implements a classifier that dispatches the packets to the appropriate μNFs. Currently, the classifier is based on matching the following 5-tuple flow signature: *(source-IP, dest-IP, ip-proto, src-port, dst-port)*.

The Tx service abstracts the NIC Tx queues and implements common functions frequently required by the μNFs. For example, in a multi-node deployment scenario, when a μNF processing graph is deployed across multiple machines, the Tx service encapsulates the packets belonging to a μNF graph destined to another machine in a custom layer 2 tunnel with appropriate tag and destination MAC addresses. The Rx service on the other end of the tunnel distributes packets to the appropriate μNFs based on the tags. These tags are determined and configured by the orchestrator.

2.7.4 Port

As discussed earlier, a port provides packet I/O abstraction for μNFs and decouples the implementation of a specific communication pattern from a μNF's packet processing logic. This design choice helps to keep the μNF implementation focused only on the packet processing part. We have two broad classes of ports, *IngressPort* for receiving packets from and *EgressPort* for sending packets to μNF(s). If not stated otherwise, ports provide a zero-copy packet exchange mechanism by exchanging the packet addresses instead of full copies of the packets. IngressPort and EgressPort present the following interfaces to the μNFs while hiding underlying implementation details: (i) pull based *IngressPort::RxBurst*, which populates an array with a burst of packet addresses; (ii) *EgressPort::TxBurst* pushes a burst of packets to the next μNF. Currently, we have the following specific implementations of IngressPort and EgressPort that allow different communication patterns between μNFs.

NIC I/O Port: A NIC I/O port abstracts the rx/tx queues in the hardware NIC. It allows μNFs to directly read from or write to the NIC. We have leveraged the NIC specific DPDK poll mode drivers (PMDs) for implementing ingress and egress versions of NIC I/O Port. The DPDK PMDs bypass the OS kernel and allow zero copy packet I/O from the NIC.

Point-to-Point Port: A point-to-point port allows a μNF to push packets to or pull packets from exactly one other μNF. We have implemented this port using a circular queue (Figure 2.7). The ingress version of the port (*PPIngressPort*) pulls a batch of packet addresses from a circular queue and the egress version (*PPEgressPort*) pushes packet addresses for a batch of packets to the queue. When a μNF's PPIngressPort and another μNF's PPEgressPort share the same circular queue, they can exchange packets with each other. The circular queue in our implementation is an instance of `rte_ring` data structure (a lock-less multi-producer multi-consumer circular queue) from DPDK `librte_ring` library.

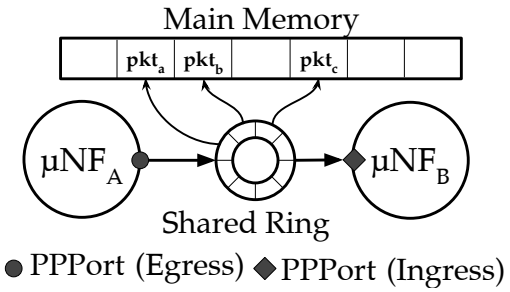


Figure 2.7: Point-to-Point Port

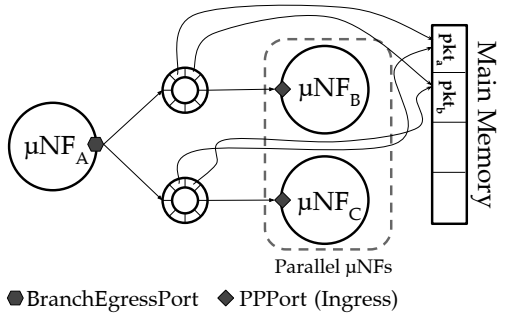


Figure 2.8: Branched Egress Port

BranchEgressPort: This port connects a μNF to multiple μNFs that are processing packets in parallel. For instance, in Figure 2.8, μNF_B and μNF_C are executing in parallel. To realize this execution model, μNF_A can be made aware of this configuration and pushes packet addresses to both of the next state μNFs. μNF_A will also need to embed the necessary meta-data in packets to mark the completion of μNF_B and μNF_C. This violates our design principle of loose coupling between μNFs, and therefore, we developed BranchEgressPort to transparently handle this type of branching. A BranchEgressPort contains multiple circular queues, each corresponding to one μNF in the next stage. Each of the circular queues can be shared with a PPIngressPort to create a communication channel. For example, one of the circular queues of μNF_A's BranchEgressPort is essentially the underlying circular queue of μNF_B's PPIngressPort. A BranchEgressPort also initializes and embeds a counter inside each packet's meta-data area, which is used to mark the completion of packet processing by all parallel μNFs.

MarkerEgressPort: A MarkerEgressPort works in conjunction with a BranchEgressPort. It is the typical EgressPort of a μNF part of a parallel processing group. This port atomically increases the embedded counter in the packet before putting the packet into a shared queue.

SyncIngressPort A SyncIngressPort connects a set of parallel μNFs to a single μNF that is potentially modifying packets. This port is also an abstraction over a shared circular queue. The queue is shared with other MarkerEgressPorts in the parallel processing group. SyncIngressPort ensures that any packet that is pulled out has been processed by all the parallel μNFs. This synchronization is done by atomically checking the counter embedded inside every packet by a BranchEgressPort. SyncIngressPort pulls a packet only if the counter value equals the number of μNFs in the parallel processing stage. In this way, the next stage of a parallel processing stage proceeds to process a packet only after all the μNFs from the parallel processing stage have completed their processing. Note that in order to keep the cost of atomically updating and checking the embedded counters, we leverage the atomic instruction set of modern CPUs.

LoadBalancedEgressPort: This is an EgressPort that load balances packets pushed by a μNF to a number of next stage μNFs. This port is particularly useful when μNFs are scaled-out. Consider two μNFs a and b, connected with a pair of ingress and egress point-to-point ports. If b is scaled out then packets from a need to be load balanced across b instances. This port transparently performs this load balancing. Our current implementation has a round-robin load balancing policy. However, more complex policies (*e.g.*, ensuring flow affinity) can also be implemented using this abstraction.

2.7.5 μNF Scheduling

In order to increase μNF density per physical machine, we share a CPU core between multiple consecutive μNFs from a μNF processing graph. This also enables these consecutive μNFs to better utilize a CPU's warm first level cache. However, like many other DPDK applications, μNFs operate in busy polling mode. Therefore, it can occur that one μNF out of several others sharing the same CPU core, gets scheduled on that core, and there is no packet at that moment to process. This will waste CPU cycles during the time allocated for the μNF. Therefore, a major challenge here is to carefully schedule μNFs to minimize the wasted CPU cycles. This is a problem of its own and merits separate investigation as seen in the literature [134]. For our prototype implementation, we aim to have a simple yet effective solution and first explore which out of the box OS scheduler is the most suitable one.

Completely Fair Scheduler (CFS) is the default scheduler in most Linux distributions [135]. CFS ensures fair sharing of a CPU between competing processes by periodically preempting them. However, there are other schedulers available in the kernel, *e.g.*, the Real Time (RT) scheduler [136]. RT scheduler supports the following two scheduling policies: First-in-First-out (FIFO) and Round Robin (RR). Unlike CFS, RT scheduler does not ensure fairness, rather

it ensures that a process only releases a CPU after it has finished (FIFO) or its allocated time quantum has expired (RR). To better understand which scheduler and scheduling policy is a best fit, we performed the following experimental study.

We deployed μNF chains of varying lengths on a single CPU core, where each μNF performs very minimal packet processing (swaps source and destination MAC addresses). We measured the throughput of these chains for smallest size (64 byte) packets using different scheduler and policy combinations, namely CFS and RT with FIFO, and RT with RR. We observed that CFS was preempting the μNFs too frequently. Consequently, there was a significant context switching overhead and μNFs from the chain were being scheduled when there was no packet available in their IngressPorts. RT scheduling was not performing well either (a few thousand packets per second throughput observed) since μNFs were getting uneven CPU time and were starving.

Therefore, we added the following optimization in the μNFs. A μNF voluntarily yields CPU in the following events: (i) when there are no packets available in its IngressPort to process, and (ii) after successfully processing k batches of packets. This optimization (voluntary yielding) improved the throughput by three orders of magnitude. We present results for different scheduler and scheduling policies with voluntary yielding optimization in Figure 2.9 and Figure 2.10.

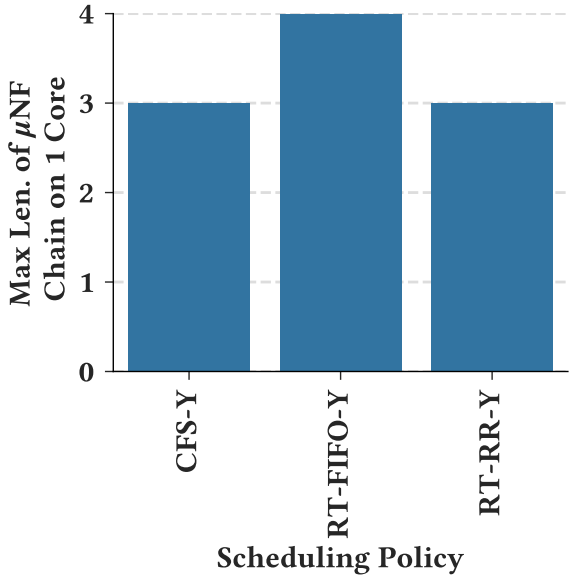


Figure 2.9: Maximum length of a μNF chain able to sustain line rate (64B) while sharing a core

Figure 2.9 shows the maximum length of a μNF chain that can be deployed on a CPU core while maintaining 10 Gbps line rate throughput for 64B packets (≈14.88 Million packets per

second (Mpps)) We found that voluntary yielding with RT scheduling and FIFO policy can support the maximum number of chained μNFs while operating at line rate.

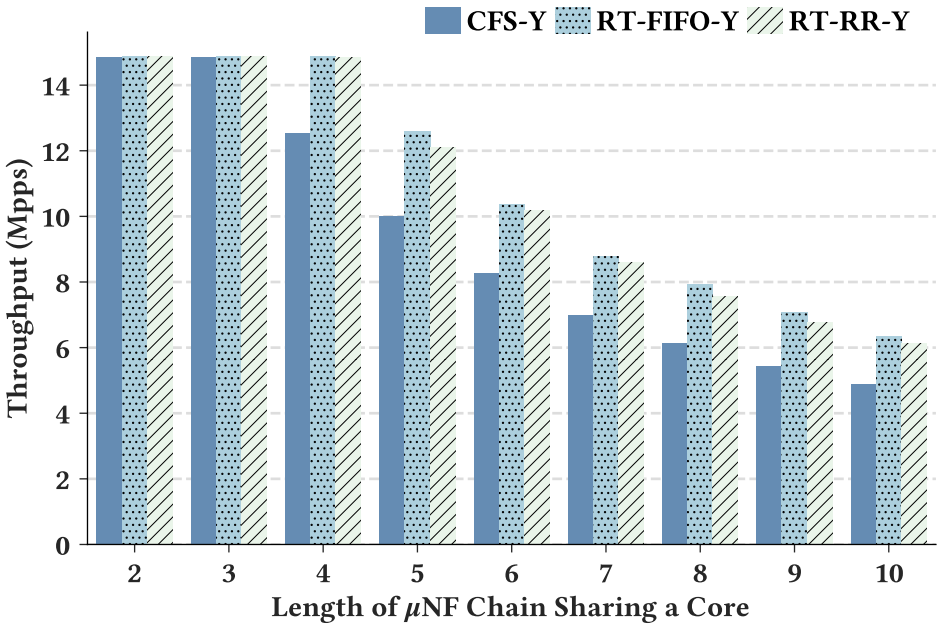


Figure 2.10: Impact of scheduler and scheduling policy on μNF chains (sharing same CPU core)

Then, in Figure 2.10 we demonstrate the throughput for μNF chains of varying lengths sharing a single core for different combinations of scheduler and policy. From our empirical evaluation, it is clear that the best combination to use is voluntary yielding with RT scheduler and FIFO policy, which is able to sustain higher throughput for any chain length compared to any of the other combinations. The reason being, CFS preempts a process as soon as its allocated time quantum expires. This means that a μNF can be preempted in the middle of processing a batch, and therefore, the next scheduled μNF is less likely to get packets from its IngressPort, thus wasting CPU cycles. RT with FIFO mitigates the impact of preempting. By combining voluntary yielding, we prevent other μNFs from starving.

2.8 Performance Evaluation

In this section, we present the results from our testbed evaluation. We first describe the experiment setup in Section 2.8.1. Then we first discuss the microbenchmark results in Section 2.8.2.

The microbenchmark results include the impact of introducing the optimizations from Section 2.6. Then, we discuss results from service level performance evaluation in Section 2.8.3. Service level performance evaluation involves evaluating and comparing finer-grain resource allocation and scaling capabilities of μNF compared to state-of-the-art system for deploying run-to-completion SFC [1] and to monolithic VNF based SFCs, respectively.

2.8.1 Experiment Setup

Hardware Configuration

Our testbed consists of two machines connected back-to-back without any switch. One of them hosts the traffic generator and the other hosts the μNFs. Each machine is equipped with 2×6-core Intel Xeon E5-2620 v2 2.1 Ghz CPU (hyper-threading disabled), 32 GB memory (evenly divided between two sockets), and a DPDK compatible Intel X710-DA 10 Gbps Ethernet adapter.

Software Environment

We used DPDK v17.05 on Ubuntu 16.04LTS (kernel version 4.10.0-42-generic). We disabled Address Space Layout Randomization (ASLR) to ensure a consistent hugepage mapping across the μNFs. We also allocated a total of 4 GB hugepages (evenly divided between sockets). Additionally, we configured the machines with the following performance improvement features:

- We isolated all CPU cores except core 0 on socket 0 from the kernel scheduler. μNF processes and agent threads were pinned to these isolated CPUs.
- CPU scaling governor was set to *performance*.
- Flow control in the NIC was disabled.

Prototype μNFs

We developed the following μNFs and used them for different evaluation cenarios:

- **MacSwapper**: Swaps the source and destination MAC address of each packet.
- **IPTtlDecrementer**: Parses IP header and decrements time-to-live (TTL) field by 1.
- **CheckIPHeader**: Computes and checks the correctness of IP checksum of each packet.

- **L3L4Filter**: Filters packets based on Layer 3-4 signature.
- **HttpClassifier**: Determines if a packet is carrying HTTP traffic by checking the payload.
- **ValidateUrl**: Performs a regular expression matching on URL in HTTP header to detect URLs containing SQL injection attacks.
- **CountUrl:** Counts the number of packets containing a certain URL in their payload.
- **ImitatedWorker:** Imitates different packet processing complexities by performing imitated processing on packets with a specified number of CPU cycles.

Traffic Generation

We used *pktgen-dpdk* [137], and *Moongen* [138] for throughput and latency measurements, respectively. We determine the physical limits of our setup by modifying the agent to receive batches of packets and echo them back (single thread pinned on a CPU core). We observed line rate throughput from this setup (*i.e.*, 10 Gbps for all packet sizes), hence, there are no bottlenecks present in the hardware or configuration. For latency measurements, we set the packet rate to 90% of maximum sustainable rate for that particular deployment scenario.

2.8.2 Microbenchmarks

Baseline Performance of μNF

We first establish the baseline performance that can be achieved by disaggregating larger VNFs into μNFs. We pinned the agent's Rx thread to a CPU core and run a very simple μNF (MacSwapper) pinned to a different CPU core in the same NUMA zone. We vary packet size from 64 to 1500 bytes and report the throughput in Figure 2.11. Throughput reaches line rate for smallest packet size on 10G NIC. We also deployed the same μNF inside a Docker container and performed the same experiment to observe any potential impact of containerization. Throughput results for containerized μNF are very similar to those presented in Figure 2.11, and are hence not presented.

Impact of Pipelined Cache Pre-fetching

We intend to utilize all available CPU cores on a machine for deploying the μNFs. However, in a NUMA system with multiple CPU sockets, processing packets on a NUMA zone other

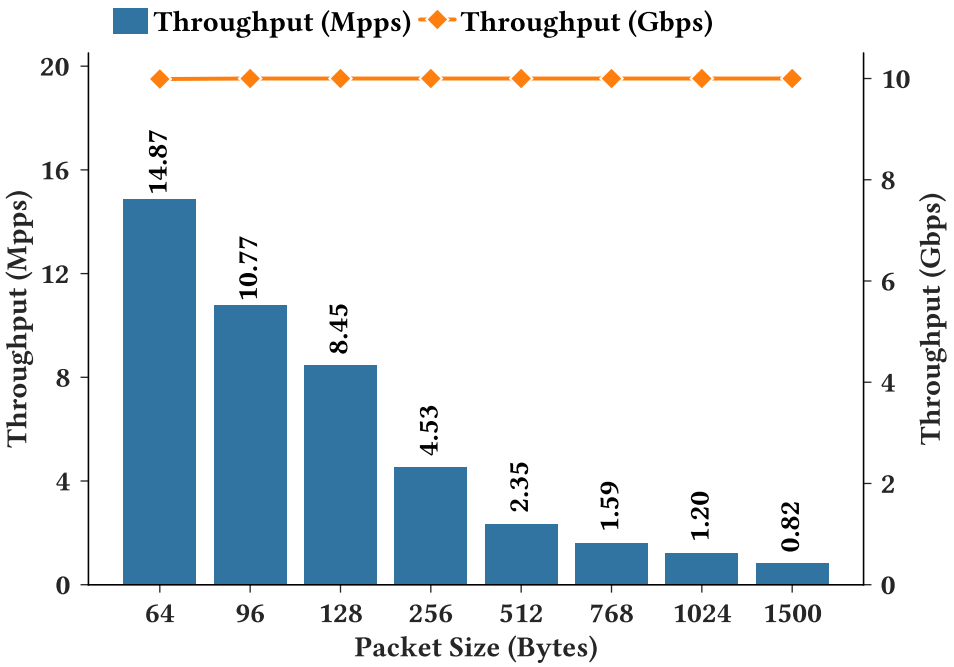


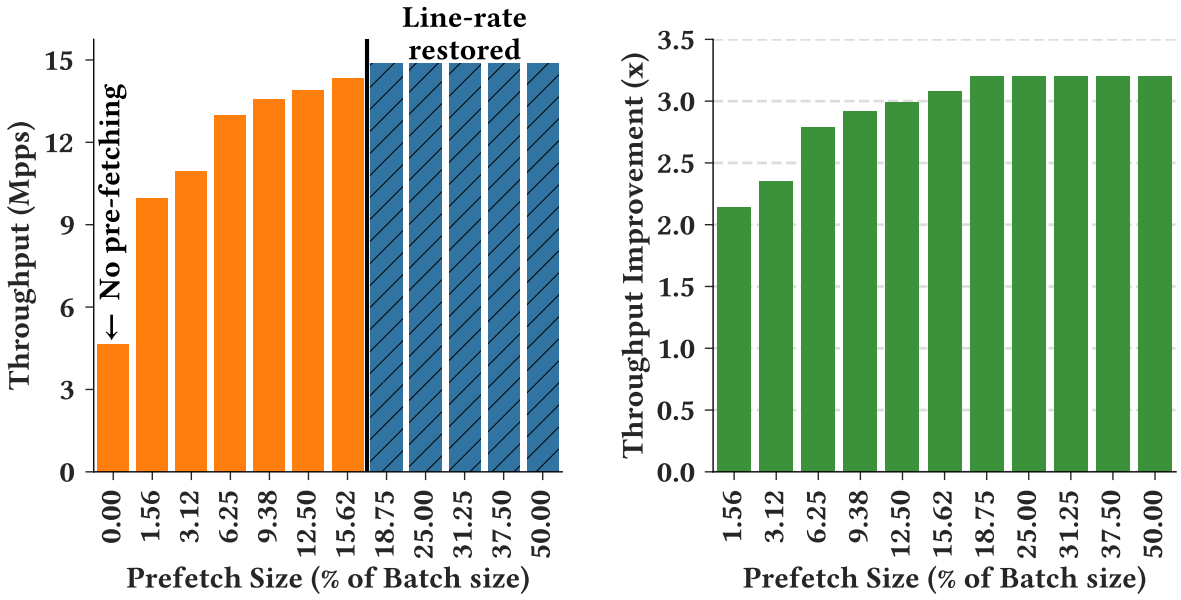
Figure 2.11: Baseline performance

than the one where the packet was received can cause performance degradation due to remote memory access overhead [139]. In this experiment, we evaluate the impact of cache pre-fetching optimization from Section 2.6.2 when packets are processed by μNFs on different NUMA nodes.

We receive packets on NUMA zone 0 and process them through a chain of two MacSwapper μNFs deployed on separate cores at NUMA zone 1. We measure throughput of this chain (for smallest size packets) while varying the number of pipelined pre-fetched packets up to 50% of packet batch size (batch size is set to 64). The results are shown in Figure 2.12. With pre-fetching disabled throughput drops to ≈30% of line rate. However, with as little as ≈20% packets pre-fetched to cache in a pipeline (8 out of 64 packets in a batch), throughput improves by more than 3× (Figure 2.12(b)) and goes back to the line rate for smallest packet size (Figure 2.12(a)).

Impact of Parallelism in μNF Processing Graph

Intuitively, parallel execution of μNFs in the processing graph is expected to reduce the processing latency for the packets through μNF processing graph. However, overheads are associated with parallel executions because of atomically increasing a counter on each packet during



(a) Throughput vs. Pre-fetch size

(b) Throughput improvement over no cache pre-fetching

Figure 2.12: Impact of pipelined cache pre-fetching

branching and synchronizing as described in Section 2.6.2. Depending on how fast a μNF is processing packets, we may observe different impacts of parallelism. To evaluate the effect of parallelism for different packet processing costs, we leveraged the ImitatedWorker μNF. We create a pipeline from four of these μNFs connected linearly for the sequential case. For the parallel case, we create a two-way branching after the first μNF (using BranchEgressPort) and join the branches at the last μNF (using SyncIngressPort). We configure the ImitatedWorker μNF to vary the per packet processing cost (in addition to what is required to swap the source and destination MAC addresses) from 0 to 700 CPU cycles. We measure packet processing latency of the sequential and parallel configurations using Moongen.

Results of this experiment (mean latency with 5th and 95th percentile error bars) are shown in Figure 2.13. When a μNF's processing cost is low (*e.g.*, less than 100 cycles/packet), the gains from parallelism are rather marginal compared to the sequential case (<10% improvement in latency). The gains become more evident when μNFs' packet processing cost increases and we see a good potential for improving latency there (*e.g.*, more than 20% for μNFs with 700 cycles per packet processing cost).

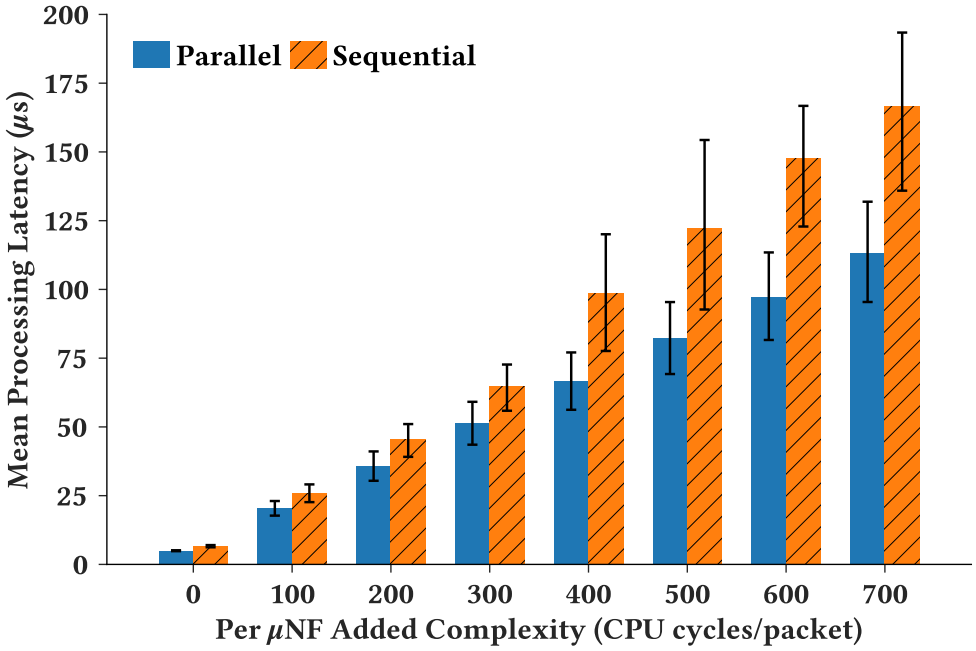
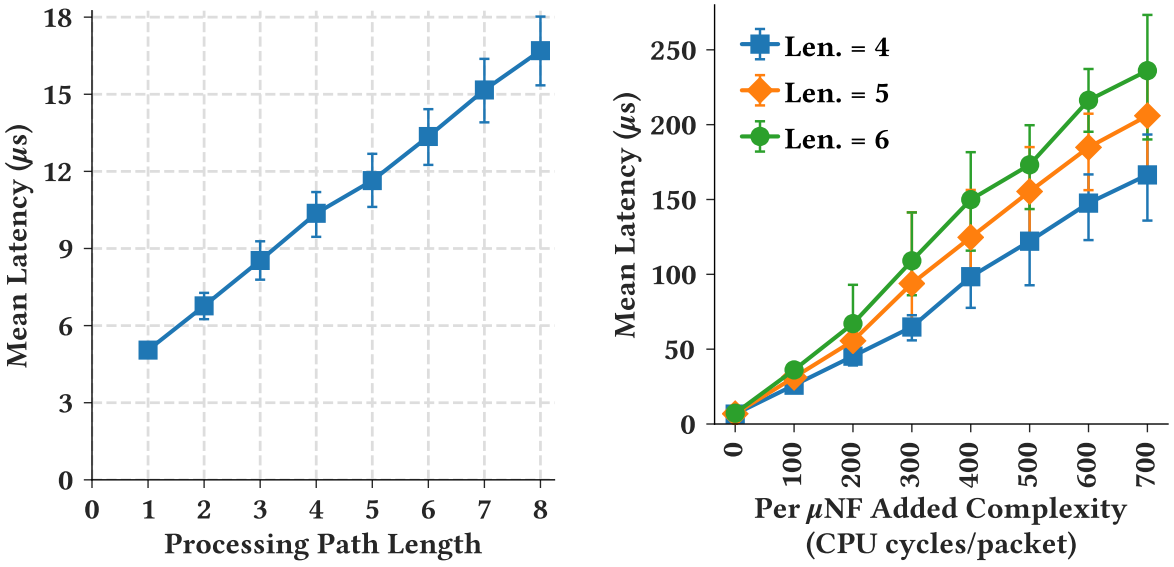


Figure 2.13: Impact of parallelism in μNF processing graph

Impact of μNF Processing Path Length

We create μNF chains of different lengths and measure packet processing latency along the pipeline using Moongen. The objective is to observe if packets start queuing up in any stage of the processing pipeline or not. We have an experiment setup similar to the scenario in Section 2.8.2. We first measure latency with varying chain lengths and without introducing any additional packet processing complexity in our MacSwapper μNF. In this case, we observe a linear increase in mean latency (Figure 2.14(a)). Then we introduce additional busy loops to emulate CPU cycles spent for packet processing (similar to Section 2.8.2) and measure latency for different lengths of μNF packet processing path (varied from 4 to 6). As we observe from Figure 2.14(b), latency increases linearly with μNF complexity as well as with μNF processing path length. Therefore, no buffering issues were encountered along the pipeline.



(a) Latency as a function of chain length

(b) Latency as a function of packet processing cost

Figure 2.14: Impact of μNF processing path length

2.8.3 Service Level Performance

Resource Efficiency over Run-to-Completion Mode

We compare μNF with NetBricks [1], the state-of-the-art in software packet processing platform operating in run-to-completion model. In particular, we perform the same experiment as in [1] to reproduce results from Fig. 7 of the original paper [1]. We developed similar packet processing element using μNF (IPTtlDecrementer) as the one used in [1] and deployed chains of different length in the following configurations:

- **NB-MC**: NetBricks with multiple threads, each pinned to a dedicated CPU core
- **NB-1C**: NetBricks with single thread
- **μNF-1C**: all μNFs packed on a single CPU core
- **μNF-MC**: the chain is divided into k clusters of consecutive μNFs such that each cluster packs maximum number of μNFs to sustain line rate while sharing a CPU core.

For a fair comparison, for both NetBricks and μNF we read packets from NIC without intervention from a software switching layer. Note that in the original paper [1], the authors spawned ℓ threads for a chain of length ℓ in NB-MC configuration and were able to reach more than 10 Gbps throughput. However, we do not have similar hardware in our disposal at this moment, hence, for each chain length, we deploy the minimum number of threads on distinct CPUs until NetBricks reaches line rate for smallest packets. We also performed the suggested performance tuning as in [1, 140].

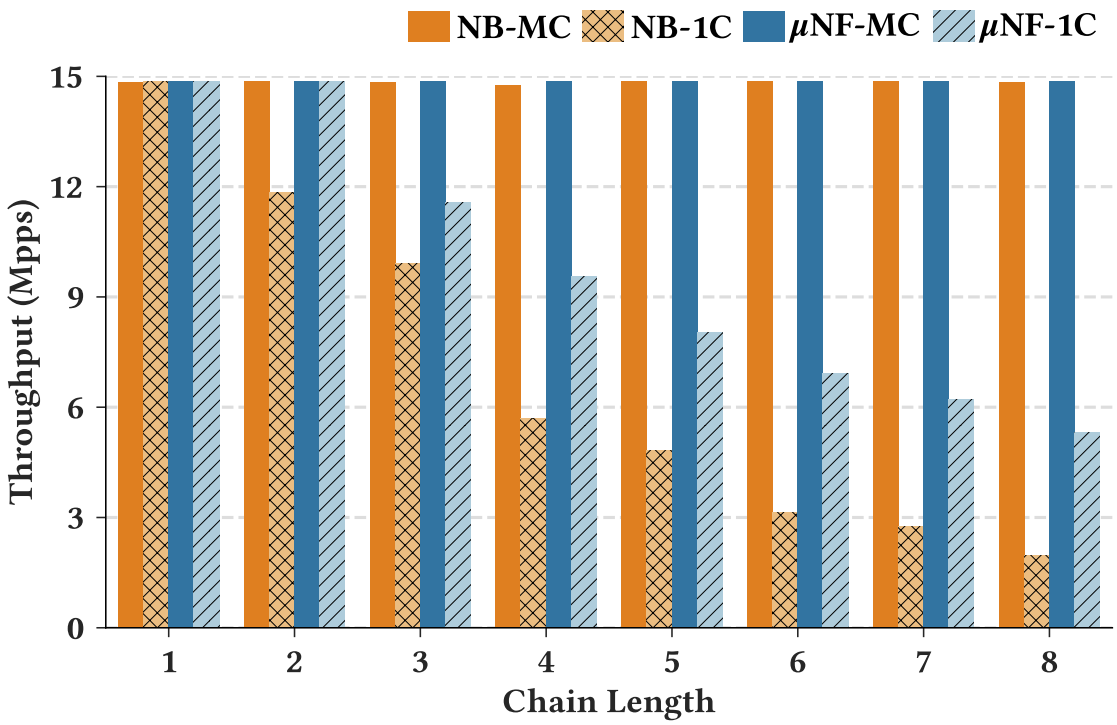


Figure 2.15: μNF vs. NetBricks [1]: throughput

The results from this experiment are shown in Figure 2.15. For the single CPU core scenario (*i.e.*, NB-1C and μNF-1C), μNF achieves better throughput than that of NetBricks with increasing chain length. Because of operating in a run-to-completion mode, NetBricks starts processing a new batch of packets only after the previous batch has finished processing through all the elements in the chain. In contrast, because of its pipeline mode, μNF can schedule a packet processing element to work on a new batch of packets as soon as that element has finished processing the previous batch and hands it over to the next element in the chain. This effectively

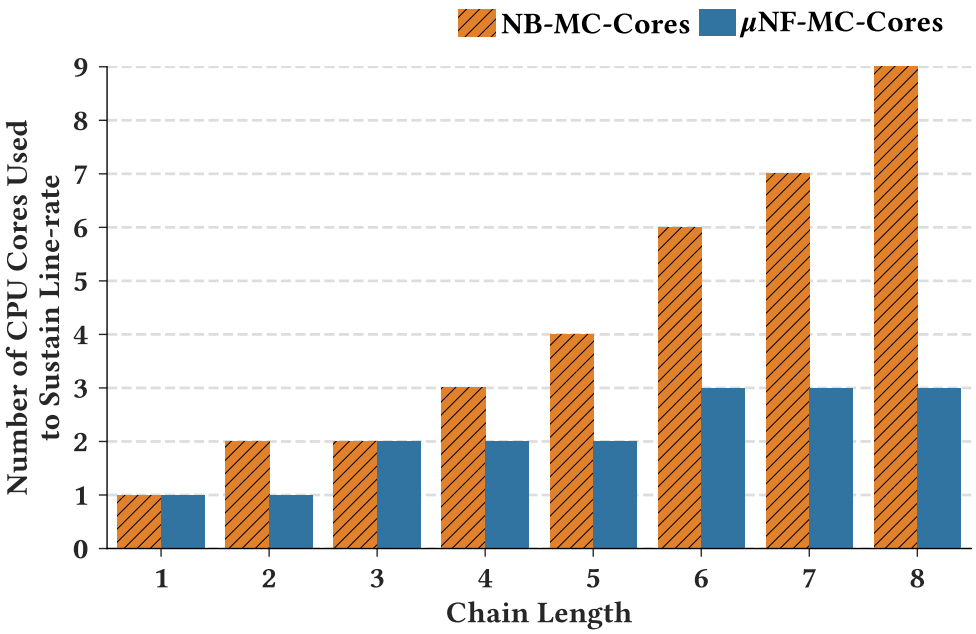


Figure 2.16: μNF vs. NetBricks [1]: number of CPU cores used

increases the number of packets in the pipeline, resulting in a better packet processing throughput compared to NetBricks as demonstrated in Figure 2.15. Indeed, there is context switching overhead involved in a pipeline mode. However, by carefully yielding the CPU as discussed in Section 2.7.5, μNF minimizes the impact of such overhead on packet processing throughput.

Then, we observe in Figure 2.16 that for a given chain length, μNF can reach line rate using lesser number of CPU cores compared to NetBricks. This is because, in pipeline mode with appropriate scheduling, it is possible to reduce wastage of CPU cycles and use the CPUs more effectively between packet processing stages, compared to run-to-completion mode. However, to be fair in the comparison, NetBricks provides packet ownership transfer by using underlying compiler features, which is not provided by μNF. Another caveat in the result is that, when we used more than 5 cores for NetBricks, the packets crossed a NUMA zone, which caused some performance penalty as we can see from the non-linear core scaling for longer chains.

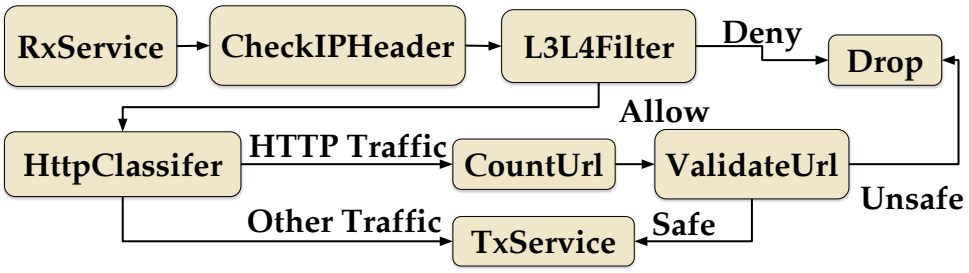


Figure 2.17: μNF realization of the SFC from Figure 2.3(a)

Table 2.2: CPU cycles saved per packet on average

Click Element Type/μNF	**CPU Cycles Saved in μNF**	**Element Weight in configuration-(i)**
CheckIPHeader	27.8%	0.44%
HttpClassifier	28.9%	47.8%
Overall	**16.8%**	–

Performance of μNF-based SFC

We have developed a set of μNFs (described in Section 2.8.1) for realizing realistic VNFs and SFCs. We use these μNFs to deploy the SFC used for the motivational experiment in Section 2.3, *i.e.*, Firewall $\rightarrow$ Monitor $\rightarrow$ Application Firewall. The resulting μNF processing graph is shown in Figure 2.17. We implemented each individual μNF as close as possible to their Click counterpart. We played the same traffic trace used in Section 2.3. Results in Table 2.2 show the relative savings in mean CPU cycles per packet when using μNF processing graph over monolithic VNFs (*i.e.*, configuration-(i) from Section 2.3). To be fair, we did not compare packet I/O from NIC since it is fundamentally different between μNF and Click. We counted the cycles spent in reading to/from ring-based shared memory since that is an added overhead in this disaggregated architecture. We also benchmarked the deployment from Figure 2.17 using pktgen. We set the packet size to 200 bytes, the average packet size reported in a recent study on a production data center [141]. Throughput reached 2.08 Mpps or 3.67 Gbps. We identified the HttpClassifier μNF to be a bottleneck through a separate benchmark. To test the scaling out of individual μNFs and LoadBalancedEgressPort, we deployed the same SFC but with two instances of HttpClassifier. We observed a near linear increase in throughput, which is 4.1 Mpps or 7.2 Gbps.

2.9 Related Works

Modular Packet Processing

The development of modular packet processing software has a long history that dates back to the late 90s. Click [116], one of the most influential works in this area proposed to build monolithic packet processing software using reusable packet processing components called *elements*. Click's focus was more on the programmability than performance. Over the years, Click influenced a significant body of subsequent research on building modular yet high performance packet processing platforms that employed different optimization techniques of their own (*e.g.*, NIC offloading, I/O batching, kernel bypass, FPGA acceleration) to improve packet processing performance and add flexibility to VNF composition [142, 143, 144, 119, 1, 145, 146]. However, these proposals are centered around the assumption that a middlebox is a monolithic software. mOS proposed to abstract abstract layer 4-7 packet processing tasks into modular and high-performance libraries for the ease of middlebox development [147]. mOS is complimentary to our work on disaggregating VNFs and can act as a facilitator for μNF development.

More recently, Slick [148] and OpenBox [48] proposed different approaches to achieve a similar goal of building packet processing from independently deployable components. Slick focuses more on the programming model for middlebox composition while OpenBox goes one step further and decouples data and control planes of VNFs. In contrast to μNF, OpenBox does not focus on addressing the engineering challenges pertaining to realizing a data plane for modular VNFs and SFCs. Its focus is more on the control aspects such as designing a protocol between VNF control and data planes, optimizing the forwarding graph, *etc.* OpenBox can complement our proposed system by acting as a control and orchestration layer above μNFs.

A chaining mechanism for lightweight VNFs has been proposed in [149], which inserts per-VM SFCs between a VM and a virtual switch for providing QoS, security, and monitoring services. In contrast, our focus is not on per-VM services, rather, on a general software architecture for realizing VNFs and SFCs from lightweight, independently deployable, and loosely-coupled packet processing components. An elaborate discussion on the challenges associated with realizing such microservice-based VNFs and SFCs can be found in [49]. An area of research orthogonal to modular and lightweight packet processing is runtime systems built around unikernels [150]. Unikernels are minimalistic OSs that are custom made to run only a single application, thus losing the benefit of being general purpose OSs. However, they have only a few megabytes of memory footprint and high deployment density (hundreds per physical machine) compared to traditional VMs or containers [119, 151], hence, can be a potential choice for μNF deployment.

Industry Efforts in Microservice-based VNFs

There has been some movement in the industry for re-designing large VNFs using microservice architecture. As part of the CORD project [5], a number of VNFs have been decomposed into having separate control and data planes that are loosely coupled and can be independently scaled. Another example is the Clearwater IP Multimedia System [152] re-architected using microservices design principle and also made available as an open-source software. However, the independently deployable components themselves are rather complex and can be further decomposed into more manageable sizes. The availability of Clearwater as an open-source software has also fostered academic research, including on enhancing its auto-scaling capabilities [153, 154], and service latency and failure recovery time [155].

Middlebox Functionality Consolidation

CoMb [47] is one of the early works to experimentally motivate the consolidation of common functionality into separate services and share them across VNFs. However, CoMb's main focus was not to address the implementation issues related to realizing such a system, rather demonstrate the advantage of consolidating multiple NFs on commodity hardware as opposed to using purpose-built hardware middleboxes. In contrast, E2 [45] proposed to consolidate management tasks such as resource allocation, fault-tolerance, monitoring, auto-scaling, *etc.* , into a single framework. More recently, Microboxes proposed to consolidate TCP protocol processing functions (*e.g.*, TCP bytestream reconstruction, TCP endpoint termination, *etc.*) of multiple middleboxes [127]. Consolidation has the advantage of reducing redundant development efforts in implementing and optimizing common tasks. In this work, we focus on the engineering efforts related to software architecture, necessary abstractions, and performance optimizations for realizing such a disaggregated packet processing platform, facilitating better consolidate of packet processing tasks with ease.

2.10 Chapter Summary

In this chapter, we described μNF, a system for building VNFs and SFCs from reusable, independently deployable, and loosely-coupled components enabling finer-grained resource allocation and scaling. Our design goal is to keep the μNFs simple and develop the necessary primitives to transparently enable different communication patterns between them. We demonstrated the effectiveness of our system through a DPDK based prototype implementation and experimental evaluation. The individual techniques used for implementing and optimizing the system are

not entirely new (*e.g.*, batched I/O, zero-copy I/O, pre-fetching). However, the bigger picture here is to demonstrate that disaggregating complex VNFs using the proposed software architecture combined with the individual techniques is indeed a viable and competitive solution for composing VNFs and SFCs. This is further supported by our experimental evaluation showing that the combined engineering effort enables finer-grained resource allocation and scaling while attaining comparable performance as state-of-the-art monolithic implementations.

Chapter 3

Multi-Layer Virtual Network Embedding

3.1 Introduction

In this chapter, we study the problem of **MU**lti-**L**ayer Virtual Network **E**mbedding (MULE) focusing on IP-over-OTN substrate networks with the objective of minimizing total resource provisioning cost for embedding a Virtual Network (VN) while considering the possibility of establishing new IP links when necessary.

Several deployment choices exist for multi-layer IP-over-Optical networks [156], including but not limited to: (i) IP-over-DWDM; (ii) IP-over-OTN-over-DWDM. DWDM networks have specific constraints such as wavelength continuity for optical circuits and typically lack transparent traffic grooming capabilities [157, 158]. A more favorable choice (also our choice of technology) is to deploy an OTN [104] over a DWDM network with advanced transport capabilities (*e.g.*, traffic grooming without optical-electrical-optical conversion). The OTN in turn can be *static*, *i.e.*, necessary interfaces on OTN nodes have been configured and the corresponding light paths in the DWDM layer have been lit to provision fixed capacity between OTN nodes. Or, the OTN can be *dynamic*, *i.e.*, more capacity can be provisioned by lighting new light paths in the DWDM layer. As a first step towards addressing VNE for multi-layer networks, we limit the scope of this chapter to the case of a *static OTN* and leave the other possible deployment scenarios for future investigation. Specifically, our contributions are as follows:

- **OPT-MULE**: An Integer Linear Program (ILP) formulation for optimally solving MULE. The state-of-the-art in multi-layer VNE does not optimally solve the problem [2]. To the best of our knowledge, this is the first optimal solution to the VNE problem for multi-layer IP-over-OTN networks.

- **FAST-MULE**: A heuristic to tackle the computational complexity of OPT-MULE. We also prove that our heuristic solves the problem optimally for a specific class of VNs, *i.e.*, star-shaped VNs with uniform bandiwdth demand.
- Evaluation of the proposed solution with with a wide-range of network sizes. Our key takeaways are: (i) FAST-MULE uses ≈1.47× more resources on average compared to the optimal solution while executing several orders of magnitude faster; (ii) FAST-MULE allocates ≈66% less resources on average compared to the state-of-the-art heuristic for multi-layer VNE [2], while accepting ≈60% more VN requests on average.

The rest of the chapter is organized as follows. We first introduce the mathematical notations representing the inputs to the problem and formally define the problem in Section 3.2. In Section 3.3, we present OPT-MULE, an ILP formulation for optimally solving MULE, followed by our proposed heuristic, FAST-MULE in Section 3.4. Our evaluation of the proposed solutions are presented in Section 3.5. We discuss the related works and contrast our solution with state-of-the-art in Section 3.6. Finally, we summerize the chapter contributions in Section 3.7.

3.2 Multi-Layer Virtual Network Embedding Problem

We first present a mathematical representation of the inputs, *i.e.*, the IP topology, the OTN topology, and the VN request. Then we give a formal definition of MULE, followed by an illustrative example.

3.2.1 Substrate Optical Transport Network (OTN)

We represent the substrate OTN as an undirected graph $\hat{G} = (\hat{V}, \hat{E})$, where $\hat{V}$ and $\hat{E}$ are the set of OTN capable devices (referred as OTN nodes in the remaining) and OTN links, respectively. Without loss of generality we assume the OTN links to be undirected, since such undirected OTN links can be either supported by specific technologies [159, 160] or by laying out multiple unidirectional fibers, or by using different wavelengths for sending and receiving within a single strand of fiber. Neighbors of an OTN node $\hat{u}$ are represented with $\mathcal{N}(\hat{u})$. We assume the OTN to be fixed, *i.e.*, light paths atop a DWDM layer have been already lit to provision OTN links $(\hat{u}, \hat{v})$ with bandwidth capacity $b_{\hat{u}\hat{v}}$. This pre-provisioned bandwidth can be used to establish IP links between IP routers. The cost of allocating one unit of bandwidth from an OTN link $(\hat{u}, \hat{v}) \in \hat{E}$ is $C_{\hat{u}\hat{v}}$. Figure 3.1 illustrates an example of an OTN network, where the numbers on each link represent its residual capacity.

3.2.2 Substrate IP Network

The substrate IP network is an undirected graph $G' = (V', E')$. Each IP node $u' \in V'$ has $p_{u'}$ number of ports with homogeneous capacity $cap_{u'}$. Each IP node u' is connected to an OTN node $\tau(u')$ through a short-reach wavelength interface. Attachment between an IP and an OTN node is represented using a binary input variable $\tau_{u'\hat{u}}$, which is set to 1 only when IP node u' is attached to OTN node $\hat{u}$. An IP link is provisioned by establishing an OTN path that connects its end points. Note that, it is common in operator networks to establish multiple IP links between the same pair of IP nodes and bundle their capacities using some form of link aggregation protocol [161]. We also follow the same practice and use (u', v', i) to represent the i-th IP link between u' and v', where $1 \leq i \leq p_{u'}$. We use the binary input variable $\Gamma_{u'v'i}$ to indicate the existence of an IP link (u', v', i) in G'. $\Gamma_{u'v'i}$ is set to 1 when IP link (u', v', i) is present in G', otherwise it is set to 0. Bandwidth of an IP link is represented by $b_{u'v'i}$. Capacity of a new IP link (u', v', i) is set to $\min(cap_{u'}, cap_{v'})$. Figure 3.1 illustrates an example IP network, where each IP link is mapped on an OTN path and the residual bandwidth capacity of an IP link is represented by the number on that link. The cost of allocating one unit of bandwidth from an IP link $(u', v', i) \in E'$ is $C_{u',v',i}$.

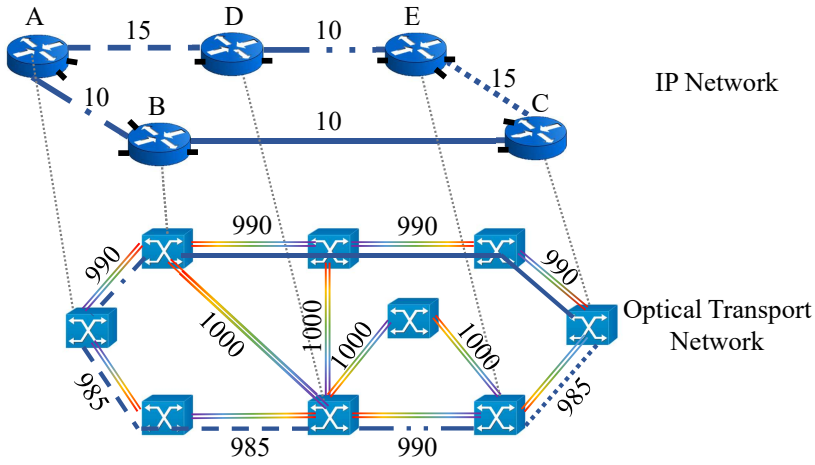


Figure 3.1: Multi-layer IP-over-OTN substrate network

3.2.3 Virtual Network (VN)

A VN request is an undirected graph $\bar{G} = (\bar{V}, \bar{E})$, where $\bar{V}$ and $\bar{E}$ are the set of virtual nodes and virtual links, respectively. Each virtual link $(\bar{u}, \bar{v}) \in \bar{E}$ has a bandwidth requirement $b_{\bar{u}\bar{v}}$.

Each virtual node $\bar{u} \in \bar{V}$ has a location constraint set $\mathcal{L}(\bar{u}) \subset V'$ that represents the set of IP nodes where $\bar{u}$ can be embedded. $\mathcal{L}(\bar{u})$ can be determined by the InP based on geographical proximity requirement by the SP. Note that $\mathcal{L}(\bar{u})$ can contain all the IP nodes to represent an unconstrained scenario. We represent the location constraints using a binary input variable $\ell_{\bar{u}u'}$, which is set to 1 if IP node $u' \in \mathcal{L}(\bar{u})$. Figure 3.2 illustrates a VN, where the number on each link represents virtual link demand, and the set next to each node denotes that virtual node's location constraints.

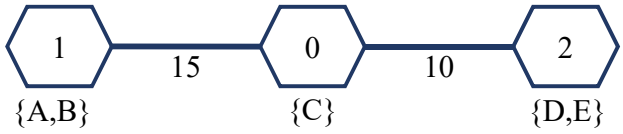


Figure 3.2: Virtual network

3.2.4 Problem Definition

Given a multi-layer Substrate Network (SN) composed of an IP network G' on top of an OTN network $\hat{G}$, and a VN request $\bar{G}$ with location constraint set $\mathcal{L}$:

- Map each virtual node $\bar{u} \in \bar{V}$ to an IP node $u' \in V'$ according to the virtual node's location constraint.
- Map each virtual link $(\bar{u}, \bar{v}) \in \bar{E}$ to a path in the IP network. This path can contain a combination of existing IP links and newly created IP links.
- Map all newly created IP links to a path in the OTN.
- The total cost of provisioning resources for new IP links and cost of provisioning resources for virtual links should be minimized subject to the following constraints:
  - IP links cannot be over-committed to accommodate the virtual links, and
  - the demand of a single virtual link should be satisfied by a single IP path.

The embedding is subject to the constraints that both IP links and OTN links cannot be over-provisioned, and virtual links and IP links cannot be routed along multiple IP paths and multiple OTN paths, respectively (*i.e.*, no path splitting). Moreover, we do not consider neither virtual

node resource requirement nor virtual node embedding cost. We assume that the virtualization enabled network devices have enough capacity to switch at line rate between any pair of ports and any complex control mechanism is decoupled and performed in a centralized control plane. Finally, we consider online version of the problem where VN requests arrive one at a time.

3.2.5 Illustrative Example

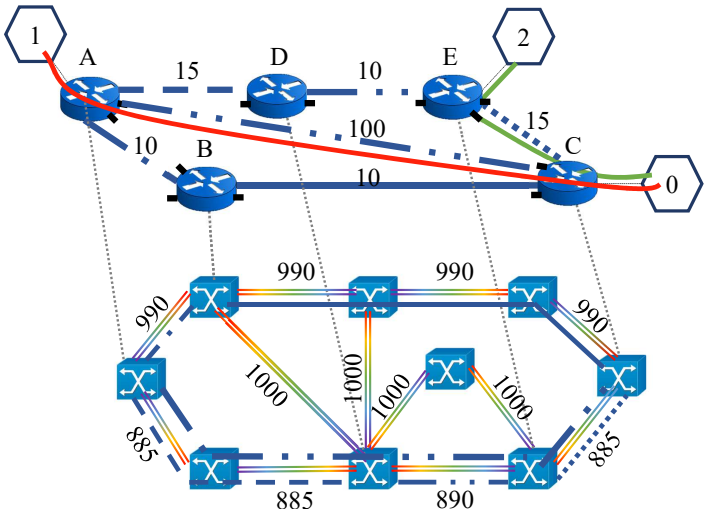


Figure 3.3: Multi-layer VN embedding example

To better illustrate the problem and the underlying complexities, consider the case of embedding the VN presented in Figure 3.2 over the multi-layer IP-over-OTN network in Figure 3.1. Given the residual capacity of the IP links, clearly, there is not sufficient bandwidth in the IP network to route the virtual link between virtual nodes 0 and 1. Hence, no feasible embedding of the VN exists. Indeed, if we were to place virtual node 1 on either IP nodes A or B, and virtual node 0 on IP node C (the only possible placement), we cannot route 15 units of bandwidth between IP nodes A and C, or B and C over an unsplittable path. In the single-layer embedding problem, such situation would have led to rejecting this VN. However, we can exploit the topological flexibility of multi-layer IP-over-OTN networks to establish new IP link(s) when there is insufficient capacity present in the IP layer.

Figure 3.3 illustrates and example solution of MULE where a new IP link has been provisioned between IP nodes A and C to accommodate the virtual link between virtual nodes 0 and 1. The new IP link is supported by provisioning resources on the OTN path between the

pair of OTN nodes connected to IP nodes A and C, respectively. This new IP link consumed an available port from IP nodes A and C. For illustration purposes, we assume uniform capacity of 100 units of bandwidth for all ports of an IP node. However, in the remainder of this chapter we do not make any assumptions on the capacity of the IP ports.

Observe that in this example, there were several possibilities for provisioning the new IP link(s). For instance, if virtual node 1 was embedded on IP node B instead of C, then the new IP link would have been between IP nodes B and C. Even when embedding virtual node 1 on IP node A, two IP links could have been created to establish a path between IP nodes A and C through B. Hence, performing the node and link embedding separately impacts the cost of the resultant embedding solution, as well as the choice of new IP link(s) to setup. This stresses the need to jointly consider both VN embedding and provisioning of new IP links.

3.3 ILP Formulation: OPT-MULE

Optimal solution to MULE needs to jointly optimize VN embedding, creation of new IP links and embedding of newly created IP links on the OTN layer. We present an Integer Linear Program (ILP) formulation for optimally solving MULE, namely OPT-MULE. We first introduce the decision variables used in our ILP (Section 3.3.1). Then we present our constraints (Section 3.3.2) followed by the objective function (Section 3.3.3).

3.3.1 Decision Variables

A virtual link must be mapped to a path in the IP network. The following decision variable indicates the mapping between a virtual link $(\bar{u}, \bar{v}) \in \bar{E}$ and an IP link, $(u', v', i) \in E'$.

$$x^{\bar{u}\bar{v}}_{u'v'i} = \begin{cases} 1 & \text{if } (\bar{u}, \bar{v}) \in \bar{E} \text{ is mapped to } (u', v', i) \in E', \\ 0 & \text{otherwise.} \end{cases}$$

Virtual node mapping on IP node is denoted by:

$$y_{\bar{u}u'} = \begin{cases} 1 & \text{if } \bar{u} \in \bar{V} \text{ is mapped to } u' \in V', \\ 0 & \text{otherwise.} \end{cases}$$

The following decision variable determines the creation of new IP links:

$$\gamma_{u'v'i} = \begin{cases} 1 & \text{when } i\text{-th IP link is created between } u' \text{ and } v', \\ 0 & \text{otherwise.} \end{cases}$$

Finally, a newly created IP link must be mapped to an OTN path. This mapping between such IP link and an OTN link is indicated by the following variable:

$$z_{\hat{u}\hat{v}}^{u'v'i} = \begin{cases} 1 & \text{if } (u', v', i) \in E' \text{ is mapped to } (\hat{u}, \hat{v}) \in \hat{E}, \\ 0 & \text{otherwise.} \end{cases}$$

In what follows, we use the notation V'^2 to denote the set of all pairs of IP nodes (u', v') such that $u' \neq v'$. A list of key notations used in the ILP formulation is presented in Table 3.1.

3.3.2 Constraints

Virtual node Mapping Constraint

Constraints (3.1) and (3.2) together ensure that each virtual node is mapped to exactly one IP node according to the location constraints. Constraint (3.3) restricts multiple virtual nodes to be mapped on the same IP Node.

$$\forall \bar{u} \in \bar{V}, \forall u' \in V' : y_{\bar{u}u'} \leq \ell_{\bar{u}u'} \tag{3.1}$$

$$\forall \bar{u} \in \bar{V} : \sum_{u' \in V'} y_{\bar{u}u'} = 1 \tag{3.2}$$

$$\forall u' \in V' : \sum_{\bar{u} \in \bar{V}} y_{\bar{u}u'} \leq 1 \tag{3.3}$$

Virtual link Mapping Constraints

Constraint (3.4) ensures that virtual links are mapped only to existing or newly created IP links. Constraint (3.5) ensures that each virtual link is mapped to a non-empty subset of IP links. We prevent the formation of loops between parallel IP links by constraint (3.6). Constraint (3.7) prevents overcommitment of IP link bandwidth. Finally, (3.8), our flow-conservation constraint,

Table 3.1: Summary of key notations

Inputs	
$\hat{G} = (\hat{V}, \hat{E})$	Substrate OTN
$b_{\hat{u}\hat{v}}$	Residual Bandwidth capacity of OTN link $(\hat{u}, \hat{v}) \in \hat{E}$
$C_{\hat{u}\hat{v}}$	Cost of allocating unit bandwidth on OTN link $(\hat{u}, \hat{v}) \in \hat{E}$ for provisioning an IP link
$G' = (V', E')$	Substrate IP Network
$\Gamma_{u'v'i} \in \{0,1\}$	$\Gamma_{u'v'i} = 1$ if $(u', v', i) \in E'$
$b_{u'v'i}$	Residual Bandwidth capacity of IP link $(u', v', i) \in E'$
$C_{u'v'i}$	Cost of allocating unit bandwidth on IP link $(u', v', i) \in E'$ for provisioning a virtual link
$p_{u'}$	Number of ports on IP node u'
$cap_{u'}$	Capacity of each port of IP node u'
$\tau_{u'\hat{u}} \in \{0, 1\}$	$\tau_{u'\bar{u}} = 1$ if IP node u' is attached to OTN node $\hat{u}$
$\bar{G} = (\bar{V}, \bar{E})$	Virtual Network Request
$b_{\bar{u}\bar{v}}$	Bandwidth requirement of virtual link $(\bar{u}, \bar{v}) \in \bar{E}$
$\mathcal{L}(\bar{u})$	Location constraint set for virtual node $\bar{u} \in \bar{V}$
$\ell_{\bar{u}u'} \in \{0, 1\}$	$\ell_{\bar{u}u'} = 1$ if $u \in \mathcal{L}(\bar{u})$, $u' \in V', \bar{u} \in \bar{V}$
Outputs	
$x_{u'v'i}^{\bar{u}\bar{v}} \in \{0, 1\}$	$x_{u'v'i}^{\bar{u}\bar{v}} = 1$ if $(u', v', i) \in E'$ is on the embedded IP path for $(\bar{u}, \bar{v}) \in \bar{E}$
$y_{\bar{u}u'} \in \{0, 1\}$	$y_{\bar{u}u'} = 1$ if $\bar{u} \in \bar{V}$ is mapped to $u' \in V'$
$\gamma_{u'v'i} \in \{0, 1\}$	$\gamma_{u'v'i} = 1$ if i-th IP link is created between u' and v'
$z_{\hat{u}\hat{v}}^{u'v'i} \in \{0, 1\}$	$z_{\hat{u}\hat{v}}^{u'v'i} = 1$ if $(\hat{u}, \hat{v}) \in \hat{E}$ is on the embedded OTN path for $(u', v', i) \in E'$

ensures that virtual links are mapped on a continuous IP path.

$$\forall(\bar{u},\bar{v}) \in \bar{E}, \forall(u',v') \in V'^2, 1 \leq i \leq min(p_{u'}, p_{v'}) : \\ x^{\bar{u}\bar{v}}_{u'v'i} \leq \gamma_{u'v'i} + \gamma_{v'u'i} + \Gamma_{u'v'i} \tag{3.4}$$

$$\forall(\bar{u},\bar{v}) \in \bar{E} : \sum_{\forall(u',v') \in V'^2} \sum_{i=1}^{p_{u'}} x^{\bar{u}\bar{v}}_{u'v'i} \geq 1 \tag{3.5}$$

$$\forall(\bar{u},\bar{v}) \in \bar{E}, \forall(u',v') \in V'^2 : \sum_{i=1}^{p_{u'}} x^{\bar{u}\bar{v}}_{u'v'i} \leq 1 \tag{3.6}$$

$$\forall(u',v') \in V'^2, 1 \leq i \leq p_{u'} : \sum_{\forall(\bar{u},\bar{v}) \in \bar{E}} x^{\bar{u}\bar{v}}_{u'v'i} \times b_{\bar{u}\bar{v}} \leq b_{u'v'i} \tag{3.7}$$

$$\forall(\bar{u},\bar{v}) \in \bar{E}, \forall u' \in V' : \sum_{\forall v' \in V'^2} \sum_{i=1}^{\min(p_{u'},p_{v'})} (x^{\bar{u}\bar{v}}_{u'v'i} - x^{\bar{u}\bar{v}}_{v'u'i}) = \\ y_{\bar{u}u'} - y_{\bar{v}u'} \tag{3.8}$$

IP Link Creation Constraints

Constraint (3.9) limits the number of incident IP links on an IP node to be within its available number of ports. Then, constraint (3.10) ensures that a specific instance of IP link between a pair of IP nodes is either decided by the ILP or was part of the input, but not both at the same time.

$$\forall u' \in V' : \sum_{\forall v' \in V' | v' \neq u'} \sum_{i=1}^{\min(p_{u'},p_{v'})} \gamma_{u'v'i} + \gamma_{v'u'i} + \Gamma_{u'v'i} \leq p_{u'} \tag{3.9}$$

$$\forall(u',v') \in V'^2, 1 \leq i \leq p_{u'} : \gamma_{u'v'i} + \Gamma_{u'v'i} \leq 1 \tag{3.10}$$

IP-to-OTN Link Mapping Constraints

First, we ensure, using constraint (3.11), that only the newly created IP links are mapped on the OTN layer. Then, constraint (3.12) is the flow conservation constraint that ensures continuity of the mapped OTN paths. Finally, constraint (3.13) is our capacity constraint for OTN links.

$$\forall (u', v') \in V'^2, 1 \leq i \leq p_{u'}, (\hat{u}, \hat{v}) \in \hat{E} : z_{\hat{u}\hat{v}}^{u'v'i} \leq \gamma_{u'v'i} \tag{3.11}$$

$$\forall (u', v') \in V'^2, 1 \leq i \leq p_{u'}, \forall \hat{u} \in \hat{V} :$$
$$\sum_{\forall \hat{v} \in \mathcal{N}(\hat{u})} (z_{\hat{u}\hat{v}}^{u'v'i} - z_{\hat{v}\hat{u}}^{u'v'i}) = \begin{cases} \gamma_{u'v'i} & \text{if } \tau_{u'\hat{u}} = 1, \\ -\gamma_{u'v'i} & \text{if } \tau_{v'\hat{u}} = 1, \\ 0 & \text{otherwise.} \end{cases} \tag{3.12}$$

$$\forall (\hat{u}, \hat{v}) \in \hat{E} : \sum_{\forall (u', v') \in V'^2} \sum_{i=1}^{p_{u'}} z_{\hat{u}\hat{v}}^{u'v'i} \times b_{u'v'i} \leq b_{\hat{u}\hat{v}} \tag{3.13}$$

3.3.3 Objective Function

Our objective is to minimize the cost incurred by creating new IP links and also the cost of provisioning bandwidth for the virtual links. Cost for provisioning new IP links is computed as the cost of allocating bandwidth in the OTN paths for every new IP link. The cost of embedding a VN is computed as the total cost of provisioning bandwidth on the IP links for the virtual links. Our objective function is formulated as follows:

$$\begin{aligned} \text{minimize} \sum_{\forall (u', v') \in V'^2} \sum_{i=1}^{p_{u'}} \sum_{\forall (\hat{u}, \hat{v}) \in \hat{E}} z_{\hat{u}\hat{v}}^{u'v'i} \times b_{u'v'i} \times C_{\hat{u}\hat{v}} \\ + \sum_{\forall (\bar{u}, \bar{v}) \in \bar{E}} \sum_{\forall (u', v') \in V'^2} \sum_{i=1}^{p_{u'}} x_{u'v'i'}^{\bar{u}\bar{v}} \times b_{\bar{u}\bar{v}} \times C_{u'v'i} \end{aligned} \tag{3.14}$$

3.3.4 Hardness of OPT-MULE

Consider the case where the IP layer has sufficient capacity to accommodate a given VN request. In this case, MULE becomes a single-layer VNE, which has been proven to be NP-Hard via a reduction from the multi-way separator problem [55]. Given that single-layer VNE is an instance of MULE, by restriction we conclude that MULE is also NP-Hard.

3.4 FAST-MULE: A Heuristic Approach

Given the NP-Hard nature of the multi-layer VNE problem and its intractability for large network instances, we propose FAST-MULE, a heuristic to solve the Multi-Layer VNE problem. We begin by explaining the challenges behind the design of FAST-MULE in Section 3.4.1, followed by a description of its procedural details and an illustrative example in Section 3.4.2 and Section 3.4.4, respectively. We analyze the running time of *FAST*-MULE in Section 3.4.3. Then, we prove in Section 3.4.5 that *FAST*-MULE yields the optimal solution for star VN topologies with uniform bandwidth requirement. Finally, we provide a guideline on how to parallelize *FAST*-MULE for leveraging multiple CPU cores (Section 3.4.6).

3.4.1 Challenges

Joint Mapping across IP and OTN Layers

One challenge of MULE is the fact that the embedding can take place in both layers. This occurs when a VN could not be accommodated by the existing IP links, and requires the creation of new ones. A plausible approach is to handle the embedding at each layer separately, *i.e.*, start by mapping the VN on the IP layer followed by mapping the new IP links on the OTN layer. Clearly, such disjoint embedding is far from optimal as there may not be sufficient bandwidth at the OTN level to accommodate the new IP links. To overcome this limitation, we equip *FAST*-MULE with the ability to consider both layers simultaneously when embedding a VN. This is achieved by collapsing the IP and OTN into a single layer graph, similar to [2]. Our collapsed graph contains all the IP and OTN nodes and links, as well as the links connecting IP nodes to OTN nodes. In contrast, [2] keeps the IP links and replaces the shortest paths in OTN with IP links that could have been created with those corresponding paths. In our case, a virtual link embedding that contains OTN links indicates the creation of new IP links.

Joint virtual node and virtual link Embedding

Another challenge is to perform simultaneous embedding of a virtual node and its incident virtual links. Embedding virtual nodes independently of their incident virtual links increases the chances of VN embedding failure. However, such joint embedding is hard to solve since it is equivalent to solving the NP-hard *Multi-commodity Unsplittable Flow with Unknown Sources and Destinations* [162]. Our goal is to equip FAST-MULE with the ability to perform joint embedding of virtual nodes along with their incident virtual links. To achieve this, we augment

the collapsed graph with meta-nodes and modify its link capacities to convert the virtual node and virtual link embedding problem into a min-cost max-flow problem that we solve using *Edmonds-Karp (EK)* algorithm [163]. The flows returned by EK indicate both the virtual nodes and virtual links mapping. In what follows, we elucidate the details of this transformation along with how the embedding solution is extracted from the flows obtained from EK.

3.4.2 Heuristic Algorithm

Algorithm 1 presents a high level view of *FAST*-MULE. From a very high level, the algorithm works as follows. First, we collapse the IP and OTN layers into a single-layer graph to perform joint optimization on both of the layers. Then, we incrementally embed the VN on the collapsed graph by extracting star subgraphs from the VN and jointly embedding the virtual nodes and virtual links of the star subgraph. We model the joint embedding problem as an instance of finding min-cost max-flow in the collapsed graph by setting appropriate flow capacities and introducing additional meta-nodes and meta-links in the collapsed graph. We describe each of the phases from Algorithm 1 in detail in the following.

Stage 1: Creation of a Collapsed Graph: We begin by collapsing the OTN and IP networks to a single-layer graph to achieve a joint embedding across both the IP and the OTN layers (*i.e.*, to address the first challenge from Section 3.4.1). The set of nodes in the collapsed graph contains all the IP and OTN nodes. The links in the collapsed graph consist of: (i) all the OTN links, (ii) added IP-to-OTN links (described later), and 3) all the IP links. We keep the residual capacities of the IP and OTN links as is. We assume the OTN links have significantly higher cost than the IP links. Therefore, new IP links are created only when they are really needed and can significantly reduce embedding cost. Finally, between every IP node u' and its corresponding OTN node $\tau(u')$, we create $p_{u'}$ links with capacity $cap_{u'}$. This guarantees that at most $p_{u'}$ new IP links can be created from u', and that their capacity cannot exceed node u''s port capacity.

Stage 2: Extraction of Star-shaped Sub-graphs from VN: Next, we randomly pick a virtual node $\bar{v} \in \bar{V}$ and embed $\bar{v}$ with its incident virtual links. Embedding $\bar{v}$'s incident links entails embedding its neighbors as well. This means that we are embedding a star-shaped subgraph of the VN at each iteration. Incremental embedding of star subgraphs was performed to jointly embed nodes and links of the VN as much as possible (*i.e.*, to address the second challenge from Section 3.4.1). To achieve this, we begin by mapping our current virtual node $\bar{v}$, *i.e.*, the center of the star to a random IP node in its location constraint set (denoted as *source* in the following). Then we construct a flow network in such a way that the paths contributing to a min-cost max-flow in the flow network correspond to the embedding of the virtual links incident to $\bar{v}$.

Algorithm 1: Multi-Layer VNE algorithm

Input: $\hat{G}$ = ($\hat{V}$,$\hat{E}$), G' = (V',E'), $\bar{G}$ = ($\bar{V}$,$\bar{E}$)
Output: Overlay Mapping Solution $\mathcal{M}$

```
1  function FAST-MULE()
2      S = {} // Initialize List of Settled Nodes
       // Step 1:  Create Collapsed Graph
3      G = CreateCollapsedGraph(G',Ĝ)
4      forall v̄ ∈ V̄ do
5          if v̄ ∈ S then continue
6          S = S ∪ v̄
           // Step 2:  Create Meta-Nodes
7          M.nmap = M.nmap ∪ MapNode(v̄,L(v̄))
8          for each (ū ∈ N(v̄)) do
9              if (ū in S) then continue
10             if (M.nmap(ū) == NULL) then V = V ∪ CreateMetaNodes(L(ū))
11             else V = V ∪ CreateMetaNodes(M.nmap(ū))
           // Step 3:  Create Ref-Nodes
12         V = V ∪ CreateRefNodes(V)
           // Step 4:  Run Link Embedding Algorithm
13         M.emap = M.emap ∪ EdmondsKarp(G)
14         E = E ∪ GetNewIPLinks(M.emap)
15         S = S ∪ isSettled(N(v̄))
16     Return M;
```

Stage 3: Addition of Meta-Nodes: We create a flow network by replacing every link in the collapsed graph with directional links in both directions. Then, $\forall \bar{u} \in \mathcal{N}(\bar{v})$, we add a meta-node in the flow network that we connect to every node in $\mathcal{L}(\bar{u})$. These meta-nodes are in-turn connected to a single meta-node, that we denote as the *sink*. After adding the meta-nodes we set the link capacities as follows:

- We set the flow capacity of a link (u, v) from the collapsed graph that is not connected with any meta-node to $\frac{b_{uv}}{\max_{\forall \bar{u} \in \mathcal{N}(\bar{v})}(b_{\bar{u}\bar{v}})}$. Setting such capacity puts an upper limit on the maximum number of virtual links that can be routed through these links. Although this can lead to resource fragmentation and in the worst case rejection of a VN, it ensures that no capacity constraints are violated.
- We set the capacity of the links incident to a meta-node to 1. This guarantees that at most $|\mathcal{N}(\bar{v})|$ flows can be pushed from *source* to *sink*.

Stage 4: Addition of Referee Nodes: Location constraint sets of different virtual nodes in a single VN may overlap. We denote such virtual nodes as *conflicting nodes* and the intersection of their location constraint sets as the *conflict set*. Every node in the conflict set is denoted as a conflict node. When conflicting virtual nodes are incident to the same *start* node, we end up with an augmented graph where all the nodes in the conflict set are connected to more than one meta-node. This is problematic because EK may end up routing multiple virtual links via the same conflict node, thereby violating the one-to-one node placement constraint. To resolve this issue, we introduce *Referee Nodes* (Ref-Nodes). Ref-Nodes are meta-nodes that are added to resolve the case of conflicting virtual nodes. In presence of a conflict, conflict nodes will be connected to more than one meta-node at the same time. Ref-Nodes are thus introduced to break this concurrency by removing the conflicting connections, and replacing them with a single connection to a Ref-node. The Ref-node is subsequently connected to all the meta-nodes of the conflicting nodes. This ensures that at most a single virtual link will be routed through any conflict node. Further, when a conflict node is selected to host a given virtual node, no other IP nodes for the same virtual node will be selected, thereby ensuring an one-to-one assignment.

Stage 5: Execution of the EK Algorithm: Now we have an instance of the max-flow problem that we will solve using the EK Algorithm. We have set the capacity of the links in the flow network in such a way that EK can push at most $|\mathcal{N}(\bar{v})|$ flows, indicating the virtual link embedding of $\bar{v}$'s incident links. Note that the only way to push $|\mathcal{N}(\bar{v})|$ flows is by having each flow traverse a unique meta-node to reach the *sink*. The virtual node embedding of $\bar{v}$'s neighbors can be extracted by examining each flow to find the incident IP node of each meta-node. If any of the obtained flows is routed via an OTN path, then a new IP link is established

and added to the collapsed graph. This allows subsequent iterations to use the newly created IP link. If at any iteration EK returns less than $|\mathcal{N}(\bar{v})|$ flows, this indicates an embedding failure, and the algorithm terminates. Otherwise, the algorithm returns to Stage 2 and repeats until all the virtual nodes are settled.

3.4.3 Running Time Analysis

We first introduce the following notations for running time analysis:

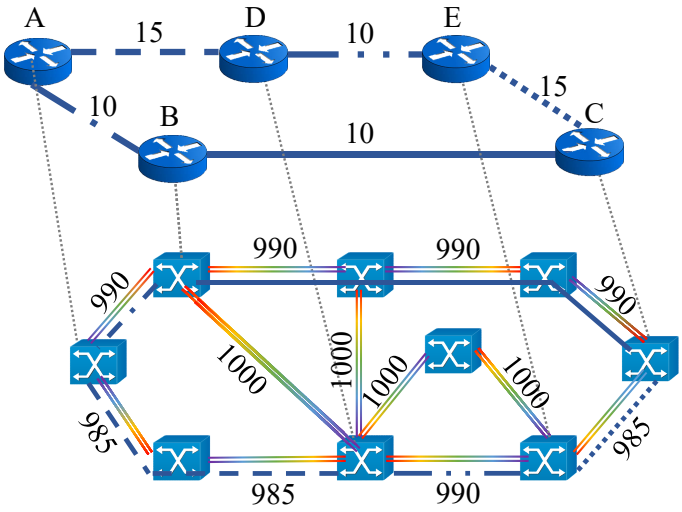
- I = the number of iterations of FAST-MULE
- $|V|$ = the number of nodes in the collapsed graph, where $|V| = O(|\hat{V}| + |V'|)$
- $|E|$ = the number of links in the collapsed graph, where $|E| = O(|\hat{E}| + |E'| + |V'|)$. The last element represents the number of IP-OTN links.

During each iteration (*i.e.*, embedding of a star subgraph from the VN), we execute the EK algorithm to find a min-cost max-flow in the collapsed graph. We replaced the augmenting path finding procedure of EK with Dijkstra's shortest path algorithm. Therefore, the running time of EK becomes $O(|V||E|^2 \log |V|)$. This renders the time complexity of our proposed approach to $O(I|V||E|^2 \log |V|)$. If we consider the worst-case scenario where the VN is in the form of a chain, and the nodes are traversed sequentially, then $I = |\bar{V}| - 1$, which results in a worst-case time complexity of $O(|\bar{V}||V||E|^2 \log |V|)$.

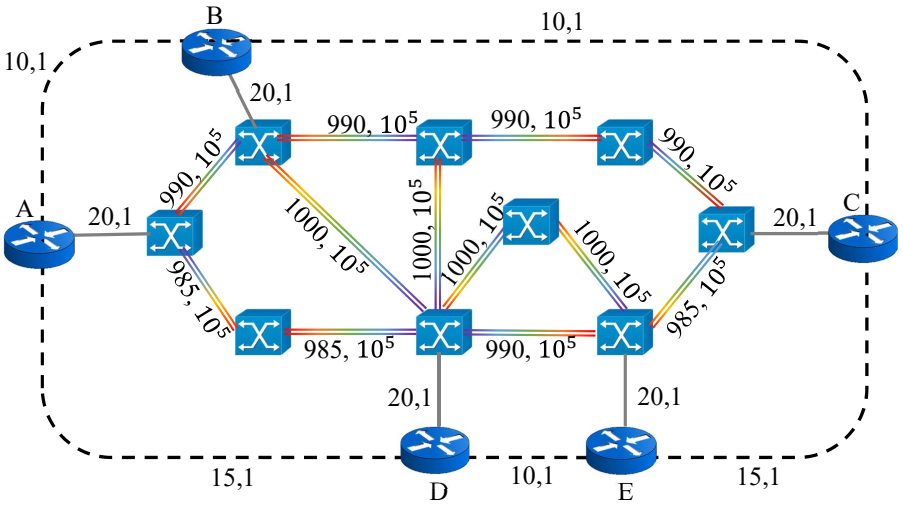
3.4.4 Illustrative Example

We use an illustrative example to describe how the heuristic finds a solution to MULE. Figure 3.4(b) shows how the IP-over-OTN graph in Figure 3.4(a) has been converted into a collapsed IP-OTN graph. The collapsed graph is composed of the OTN nodes, OTN links, IP Nodes, IP-OTN links (represented by the single straight gray lines), and IP links (represented by the dashed black lines). Here, we assume that each IP node has a single residual port of capacity 20. The numbers on each link represent the capacity of the link followed by the cost of using this link. Observe that we set the cost of the IP links to 1, whereas the cost of the OTN links is set to a really high number to discourage the routing from passing through OTN links.

Next, we showcase how *FAST*-MULE embeds the VN in Figure 3.2 atop the collapsed graph, as illustrated in Figure 3.5. We consider that virtual node 0 is the start node. Hence, the source



(a) IP-over-OTN Graph



(b) Collapsed IP-OTN Graph

Figure 3.4: Transformation from multi-layer to single-layer substrate network

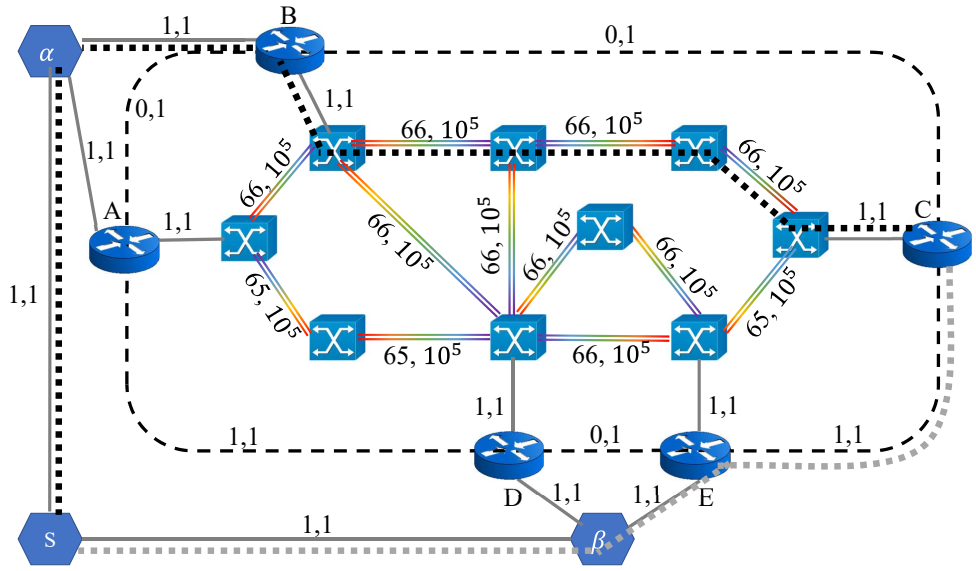


Figure 3.5: FAST-MULE: an illustrative example

node at this iteration of EK is IP node C. The sink node is meta-node s attached to the meta-nodes α and β of virtual nodes 1 and 2, respectively. Given that the maximum demand of virtual node 0's incident links is 15, the capacity of each link in the collapsed graph (except links incident to meta-nodes whose capacity is fixed to 1) is replaced by the number of virtual links of capacity 15 it can accommodate. Running EK on the augmented graph (Stage 5) returns two flows between the source node C and the sink node s, indicated by the black and grey dotted lines in Figure 3.5. Here, we observe that EK can only route virtual link (0,2) via existing IP links (grey flow); whereas virtual link (0,1) is routed through OTN links (black flow), thereby creating a new IP link (B,C) with capacity 20. Further, by examining the terminating IP nodes in every flow, we identify the virtual node embedding of nodes 1 and 2 as IP nodes B and E, respectively.

3.4.5 Optimality of FAST-MULE for Star VN Topology

Recall that in Algorithm 1, the joint node and link embeddings are executed iteratively on a subgraph of the VN until all the virtual nodes are settled. This iterative scheme renders a sub-optimal solution. However, if we could perform a joint node and link embedding on the entirety of the VN in a single iteration, that would guarantee that the obtained solution is indeed optimal. Such embedding is possible when all the nodes in the VN are only connected to a single node,

and if the latter is selected as the start node, *i.e.*, the VN topology is a star. A star VN topology $S(N)$ contains a center node $\bar{u}$ and N links connecting $\bar{u}$ to N leaf nodes $\{\bar{v}_1, \bar{v}_2, \ldots \bar{v}_N\}$. In the sequel, we prove that Algorithm 1 can find the optimal solution in polynomial time when the VN request is a star topology (typically used to support multi-cast services [58]) with identical bandwidth demand on all virtual links.

Theorem 1. *Given a star VN topology $\bar{G} = S(N)$ with uniform bandwidth demand β for all virtual links, Algorithm 1 obtains the optimal solution in polynomial time.*

Proof. The optimal embedding of $\bar{G}$, $\mathcal{M}^*$, is the one where the virtual nodes are placed on the IP nodes that provide the lowest cost link embedding. The cost includes both the cost of provisioning new IP links and the cost of allocating bandwidth for virtual links. We denote the cost of $\mathcal{M}^*$ as $\theta^* = \beta \sum_{i=1}^{N} \sum_{u'v' \in P_{\bar{u}\bar{v}_i}} C_{u'v'}$, where $P_{\bar{u}\bar{v}_i}$ is the embedding path for virtual link $(\bar{u}, \bar{v}_i)$. Without loss of generality, we abstract a newly created IP link (u', v')'s cost as $C_{u'v'}$. Let $\mathcal{M}$ be the solution obtained by Algorithm 1. For simplicity, we assume the central node $\bar{u}$ has exactly one IP node in its location constraint set. $\mathcal{M}$ consists of placing $\bar{u}$ on the IP node in its location constraint set, v', followed by running EK from v' to the sink node s. EK will return the min-cost max-flow from v' to the sink node s. Given that the capacity of all the incident links to s are 1, the number of flow augmenting paths will be at most the number of leaf nodes in $\bar{G}$ and exactly 1 unit of flow will be pushed through each of these augmenting paths. Therefore, upon successful embedding, EK will return N flow augmenting paths with minimum cost θ. Now recall that the only way to push N flows towards the sink is to traverse every meta-node once; which entails the traversal of one node from each location constraint set. The traversed nodes represent the virtual node embedding of all the leaf nodes in $S(N)$. Therefore, the flow augmenting paths represent a valid embedding of $S(N)$. We can characterize θ as, $\theta = \sum_{i=1}^{N} \sum_{(u,v) \in F_i} C_{uv} \times f_{uv}$, where F_i is the i-th flow augmenting path and f_{uv} is the flow pushed along link (u, v) in the flow network constructed from the collapsed graph. Note that, $f_{uv} = 1$, therefore, the cost becomes, $\theta = \sum_{i=1}^{N} \sum_{(u,v) \in F_i} C_{uv}$. If we can prove that $\sum_{i=1}^{N} \sum_{(u,v) \in F_i} C_{uv} = \sum_{i=1}^{N} \sum_{u'v' \in P_{\bar{u}\bar{v}_i}} C_{u'v'}$ then our proof is complete. Since θ^* is the optimal objective value, let, $\sum_{i=1}^{N} \sum_{(u,v) \in F_i} C_{uv} > \sum_{i=1}^{N} \sum_{u'v' \in P_{\bar{u}\bar{v}_i}} C_{u'v'}$. Then it implies that if we pushed the flows along the paths $\bigcup_{i=1}^{N} P_{\bar{u}\bar{v}_i}$ (the newly created IP links can

be expanded to a set of OTN links to match the paths in the collapsed graph), we would have obtained a lower cost solution to min-cost max-flow problem, which contradicts that θ is the minimum cost of our min-cost max-flow problem for the converted flow network. Therefore, $\sum_{i=1}^{N} \sum_{(u,v)\in F_i} C_{uv} = \sum_{i=1}^{N} \sum_{u'v'\in P_{\bar{u}\bar{v}_i}} C_{u'v'}$, completing our proof. □

If the central node, $\bar{u}$, has more than one candidate node in its location constraint set, then running Algorithm 1 $|\mathcal{L}(\bar{u})|$ times is sufficient to obtain the lowest cost mapping solution, and the running time of Algorithm 1 still remains polynomial.

3.4.6 Parallel Implementation of FAST-MULE

Note that during the execution of stage 2 in FAST-MULE, *i.e.*, during the extraction of star-subgraphs from the VN, we randomly chose a virtual node as the center node of the extracted star graph. Indeed, the order in which the virtual nodes are chosen for star subgraph extraction has an impact on the performance of the heuristic. Therefore, we propose to execute the heuristic for a set of virtual node orderings and choose the least cost one from the resulting solutions. Clearly, this means increasing the order of complexity for FAST-MULE.

One way to consider different virtual node ordering in FAST-MULE is to consider the different virtual node orders in parallel, *i.e.*, implement FAST-MULE as a multi-threaded program to utilize the multiple CPU cores on modern machines. Each thread of execution computes a solution to MULE by taking a virtual node ordering as an input. Since, one execution of FAST-MULE for one virtual node ordering is independent of another execution with a different virtual node ordering, therefore, they can be run in parallel without requiring any synchronization between the threads. After the parallel executions finish, we can choose the best embedding, *i.e.*, the least cost embedding among all the parallel executions.

3.5 Evaluation Results

We evaluate our proposed solutions for MULE through simulations. Due to the lack of publicly available real world multi-layer network topologies, we resort to generating synthetic topologies with varying sizes for our performance evaluation. We first describe our simulation setup in Section 3.5.1 and the evaluation metrics in Section 3.5.2. Then we present our evaluation results based on the following two scenarios: (i) micro-benchmarking of FAST-MULE by comparing with the optimal solution and to D-VNE [2], the state-of-the-art heuristic for solving

multi-layer VNE problem (Section 3.5.3), and (ii) steady-state analysis of the performance of FAST-MULE and comparison with that of D-VNE [2] (Section 3.5.4).

For the micro-benchmarking scenario, we consider the VN requests in isolation, assuming each VN request can be successfully embedded on the SN. Micro-benchmarking allows us to measure how resource efficient is FAST-MULE compared to the optimal solution and to D-VNE [2]. In contrast, for the steady-state scenario, we consider VN arrival and departure over a period of time and consider the possibility of failing to embed VN requests on the SN. The steady-state analysis gives insights on substrate resource utilization over a longer period of time.

3.5.1 Simulation Setup

Testbed

We have implemented OPT-MULE and FAST-MULE using IBM ILOG CPLEX 12.5 C++ libraries and Java, respectively. OPT-MULE was run on a machine with 4×8 core 2.4 Ghz Intel Xeon E5-4640 CPU and 512 GB of memory, whereas, we used a machine with 2×8 core 2 Ghz Intel Xeon E5-2650 CPU and 256 GB memory to evaluate FAST-MULE. We used a home-grown discrete event simulator to simulate the arrival and departure of VNs for the steady state scenario.

Multi-Layer IP-over-OTN Topology

As mentioned earlier, due to the lack of publicly available real multi-layer network topologies, we resorted to synthetically generating the multi-layer SN topologies. For the micro-benchmarking scenario, we generated OTNs by varying the size between 15–100 nodes. For each OTN, we generated an IP topology with a node count of 60% of that of the OTN. Each node in the IP topology was attached to exactly one node in the OTN topology. For both the OTN and the IP topologies, we set a link generation probability to match their average node degree to known ISP topologies [164]. For the steady state scenario, we generated a larger SN topology with a 150 node OTN and 90 node IP network. Choice of such a size is based on the average size of known ISP networks found in the literature [164]. The link generation probability was again chosen to ensure node degrees are similar to known ISP topologies. For both scenarios, OTN links were assigned a capacity of 100 Gbps, while IP links were assigned a capacity randomly chosen between 10–20 Gbps. Finally, we used a constrained shortest-path algorithm to map the input IP links over OTN paths.

VN Topology

For the micro-benchmarking scenario, we generated 20 VNs for each combination of IP and OTN, each VN with 4–8 virtual nodes. For the steady state case, we varied the size of the VN between 4–15 virtual nodes. For both scenarios, we set a 50% probability of having a link between every pair of virtual nodes. virtual link capacities were randomly set between 50%–100% of that of the IP links. For each virtual node, we generated a location constraint set by randomly selecting an IP node and including all the IP nodes withing its 3-hop reach. For random graph generation (both the VN and the SN) we used Erdos – Renyi method [165].

We evaluated the arrival and departure of VNs in the steady state scenario by simulating a Poisson process. We varied the VN arrival rate between 4 to 10 VNs per 100 time units, with a VN lifetime exponentially distributed with a mean of 1000 time units. These chosen set of parameters conforms with the ones used in the research literature [55, 166, 167].

3.5.2 Evaluation Metrics

- **Cost Ratio** This is the ratio of costs obtained by two different approaches for solving the same problem instance, where cost is computed using (3.14). Cost ratio measures the relative performance of two approaches.
- **Execution Time** The time required for an algorithm to solve one instance of MULE.
- **Acceptance Ratio** The fraction of VN requests that have been successfully embedded on the SN over all the VN requests.
- **Utilization** The Utilization of an IP link is computed as the ratio of total bandwidth allocated to the embedded virtual links to that IP link's capacity
- **Embedding Path Length** The length of IP (or OTN) path corresponding to a virtual link's (or new IP link's) embedding.

3.5.3 Micro-benchmarking Results

We focus our micro-benchmarking on the following aspects: (i) cost comparison between FAST-MULE and OPT-MULE to evaluate how well FAST-MULE compares to the optimal, (ii) impact of virtual node ordering on FAST-MULE's performance, and (iii) comparison of FAST-MULE with the state-of-the-art heuristic [2] for solving multi-layer VNE problem.

Optimality Gap of FAST-MULE

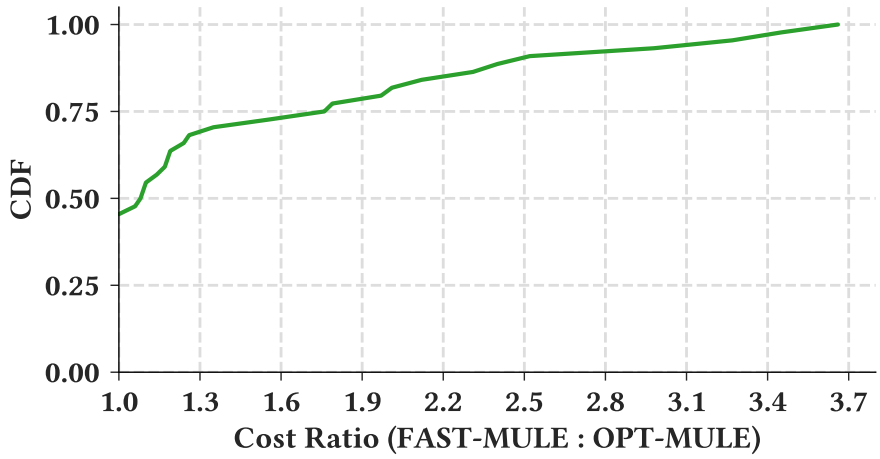


Figure 3.6: FAST-MULE to OPT-MULE cost ratio

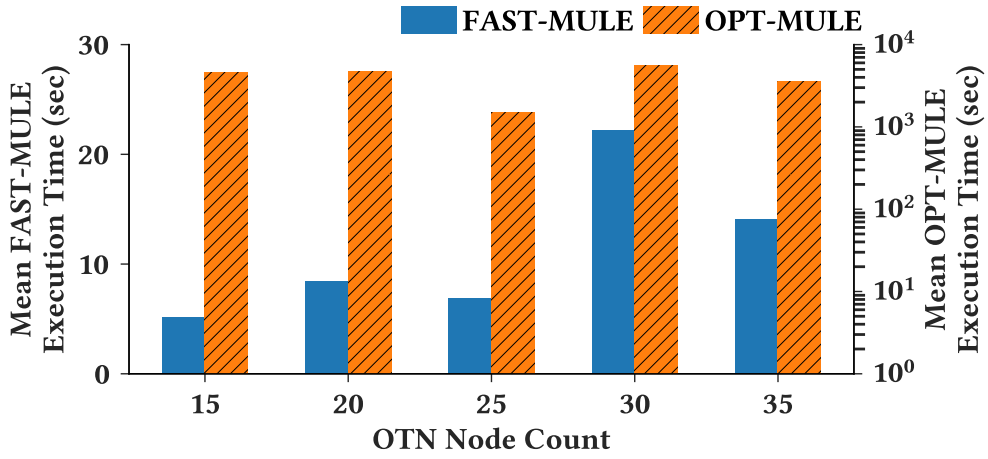


Figure 3.7: Comparison of execution time

First, we empirically measure the extent of additional resources allocated by FAST-MULE compared to OPT-MULE. Our cost function is proportional to the total bandwidth allocated for a VN and the new IP links. Therefore, cost ratio of FAST-MULE to OPT-MULE gives the extent

of additional resources allocated by FAST-MULE. Figure 3.6 shows the Cumulative Distribution Function (CDF) of cost ratio between FAST-MULE and OPT-MULE. Note that, OPT-MULE scaled up to only 35-node OTN. To mitigate the impact of virtual node ordering during embedding, we run FAST-MULE 75 times, each time with a different virtual node embedding order and take the best solution at the end. We observe from the results that 50% of the VNs admitted by FAST-MULE have an embedding cost within 10% of the optimal solution. On average, the admitted VNs have a cost within 1.47× of that of the optimal solution. These results are indeed promising given that FAST-MULE achieves this while executing 440× faster than OPT-MULE on average (10s for FAST-MULE vs. >1hr per VN for OPT-MULE).

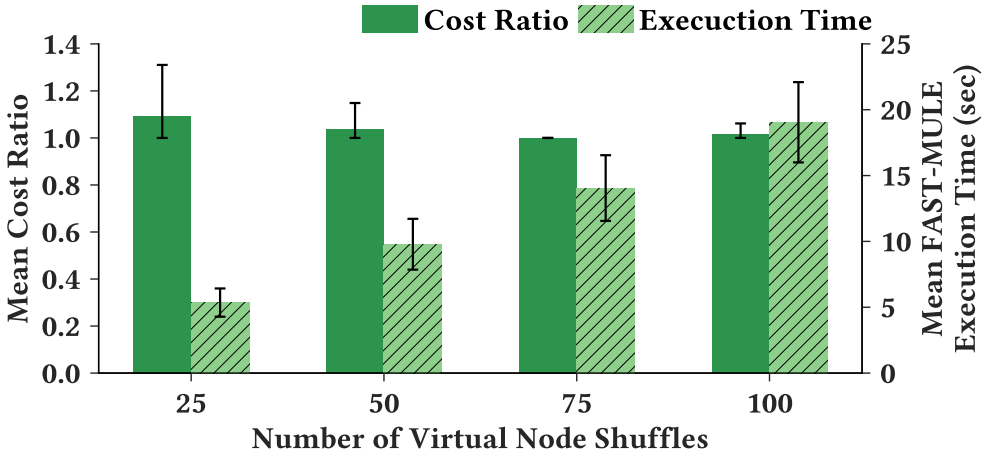


Figure 3.8: Impact of virtual node shuffle on FAST-MULE's performance

To further showcase the advantage of FAST-MULE compared to OPT-MULE we plot their execution times against varying SN size in Figure 3.7. For similar problem instances in our evaluation, FAST-MULE executed 200 – 900 times faster than OPT-MULE. Even after increasing the SN size, the execution time of FAST-MULE remained in the order of tens of seconds.

Trade-off between Cost Ratio and Execution Time

We also evaluated the impact of the number of virtual node orderings considered for the embedding. We present the results in Figure 3.8, which shows how increasing the number of considered virtual node orderings impacts the cost ratio and the execution time of FAST-MULE. Clearly, as we increase the number of considered virtual node orderings, FAST-MULE to OPT-MULE cost ratio decreases. This comes at the expense of increased execution time, which still

remains in the order of tens of seconds. However, the gain becomes marginal as we go beyond 75 iterations. Hence, in our evaluation we opt for feeding FAST-MULE with 75 virtual node orderings and select the best solution.

Comparison of FAST-MULE with D-VNE [2]

Now, we evaluate how well FAST-MULE performs compared to the state-of-the art heuristic for multi-layer VNE [2]. We refer to [2] by D-VNE in the remaining. D-VNE constructs an auxiliary graph from the IP and Optical layers. The auxiliary graph contains precomputed optical paths that can be potentially chosen for creating new IP links. In contrast, we do not precompute paths in the OTN layer and let the embedding decide the best set of paths for jointly embedding virtual links and possible new IP links. D-VNE first embeds the virtual nodes using a greedy matching approach and then uses shortest path algorithm to route the virtual links between embedded virtual nodes. We modified D-VNE to fit to our context where we do not perform wavelength allocation and omit node resource requirements.

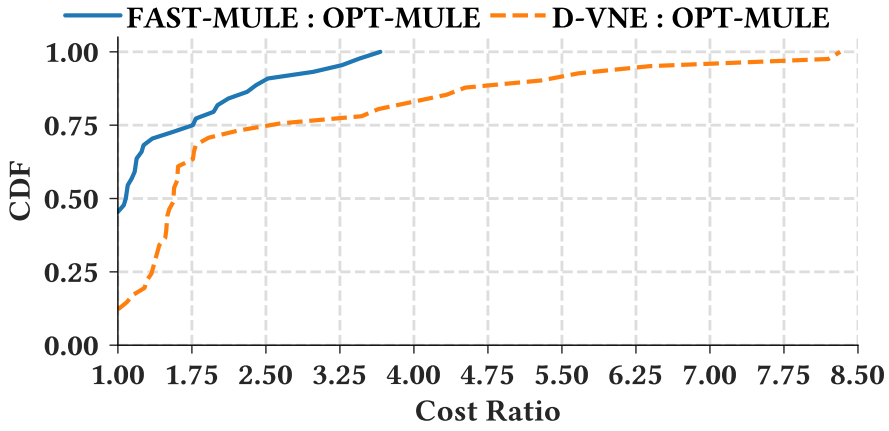
We begin by evaluating the cost ratio of D-VNE to OPT-MULE (Figure 3.9(a)). The performance gap between D-VNE and FAST-MULE is evident from Figure 3.9(a). D-VNE could embed VNs within 1.5$\times$ the cost of the optimal for $\approx$40% of the cases, whereas, FAST-MULE remains within the same bound for more than 70% of the cases. A head-to-head comparison between D-VNE and FAST-MULE is presented in Figure 3.9(b). We observe that on average D-VNE allocates $\approx$70% more resources compared to FAST-MULE. These results reflect the advantage of a joint embedding scheme compared to a disjoint approach adopted by D-VNE.

3.5.4 Steady State Analysis

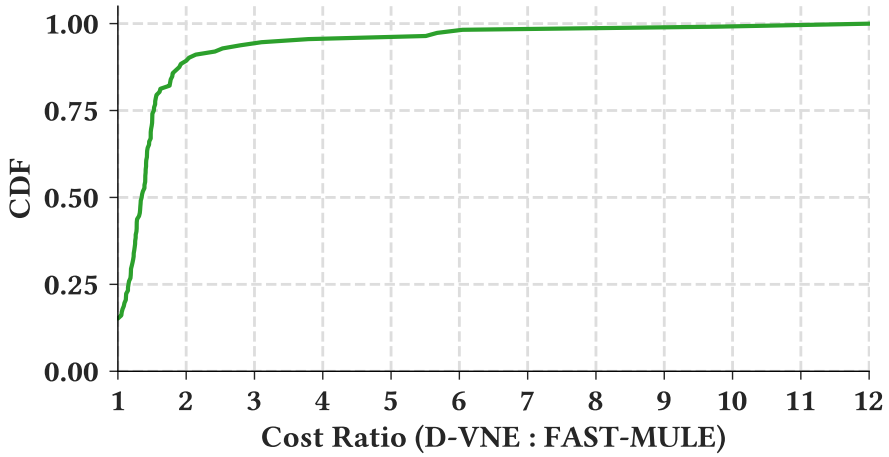
We perform a steady state analysis using the VN arrival rate and duration parameters described in Section 3.5.1 for a total of 10000 time units. The total number of VNs across the simulations were varied between 400 - 950. The steady state performance analysis is focused on the following aspects: (i) comparing the acceptance ratio obtained by FAST-MULE to that of D-VNE under different loads, (ii) analyze and compare the load distribution on the SN, and (iii) analyze topological properties of the solution.

Acceptance Ratio

In this section, we present results on the acceptance ratio obtained by FAST-MULE and compare that with the acceptance ratio obtained by using D-VNE [2]. We consider the first 1000 time



(a) Comparison between D-VNE and OPT-MULE



(b) D-VNE to FAST-MULE Cost Ratio

Figure 3.9: Comparison between D-VNE, OPT-MULE, and FAST-MULE

units of the simulation as the warm up period and discard the values from this duration. We compute the mean of the acceptance ratio obtained during the rest of the simulation period and report it along with the 5th and 95th percentile values against different VN arrival rates in Figure 3.10. The results show that FAST-MULE outperforms D-VNE in all cases and accepts at least $\approx$37.5% more VNs over all cases. When the system load is increased, *i.e.*, for higher VN arrival rate, this gap is even bigger. For instance, when VN arrival rate is 10 VNs per 100 time

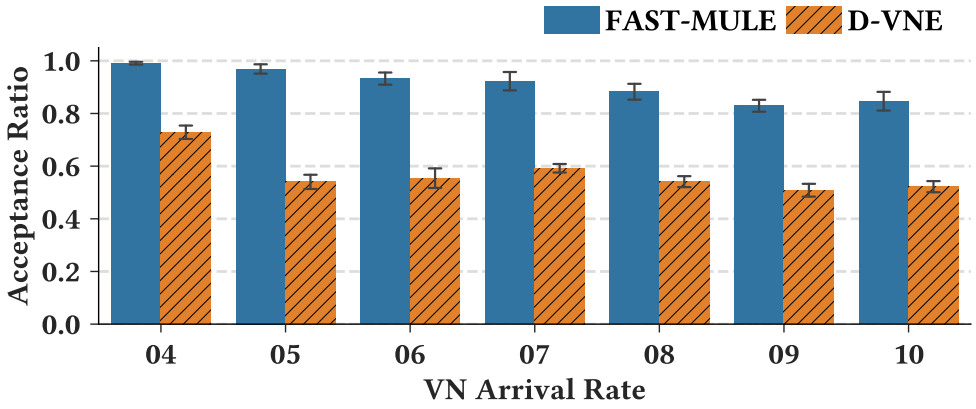


Figure 3.10: VN acceptance ratio

unit, FAST-MULE accepts ≈78% more VNs compared to D-VNE.

There can be several contributing factors to such behavior. For instance, one possibility is that the SN is becoming saturated (due to sub-optimal embedding with longer embedding paths), hence, VN requests are being rejected more often. Another possibility is that sufficient capacity in the SN is available, however, an algorithm is unable to exploit the available capacity or exploit the topological flexibility offered by the multi-layer network. In the following, we analyze how these algorithms distribute load over the substrate and how much they are able to exploit the topological flexibility to gain further insight into the difference in acceptance ratio.

Load Distribution

We measure the utilization of IP links at each VN arrival and departure event. We present the mean IP link utilization for varying load (*i.e.*, VN arrival rate) in Figure 3.11. One interesting observation is that, although D-VNE yields a lower acceptance ratio, it exhibited a higher mean link utilization compared to FAST-MULE (≈10% more). However, this plot does not capture the variance in link utilization and how the load is distributed over the IP links.

We present a further break down of IP link utilization in Figure 3.12. Specifically, we present the Cumulative Distribution Function (CDF) of the mean, the 5th percentile, and the 95th percentile link utilization in Figure 3.12(b), Figure 3.12(a), and Figure 3.12(c), respectively. The results for load distribution is also consistent with that from Figure 3.11, *i.e.*, D-VNE consistently exhibits higher link utilization while yielding accepting less VNs compared to FAST-MULE.

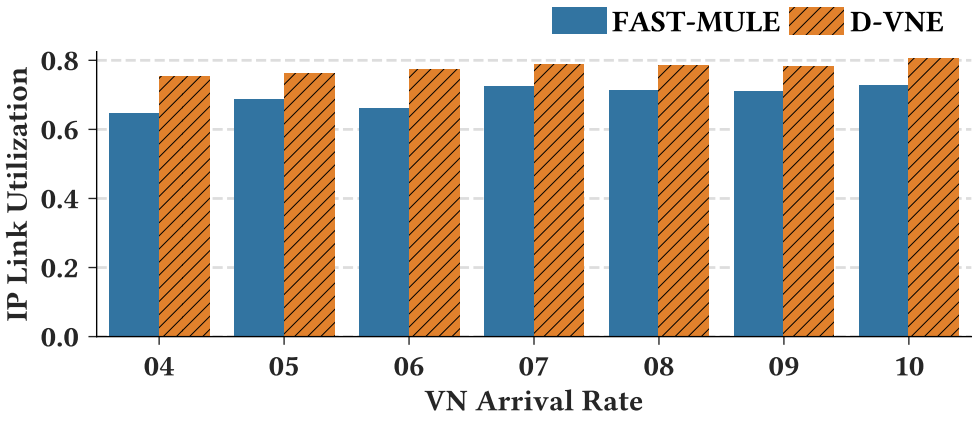
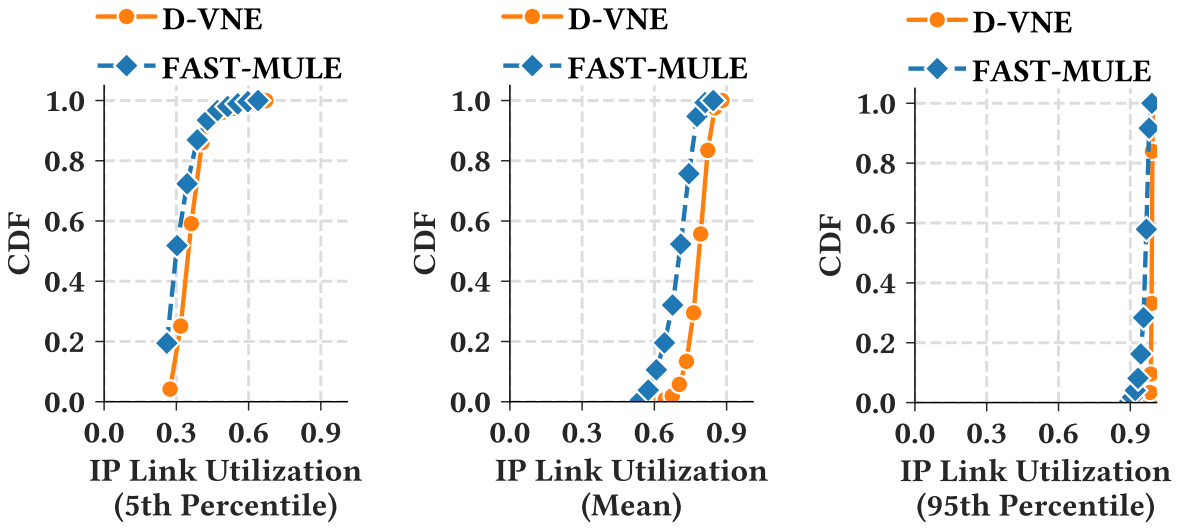


Figure 3.11: Mean IP link utilization with varying load



(a) CDF of 5th percentile IP link utilization

(b) CDF of mean IP link utilization

(c) CDF of 95th percentile IP link utilization

Figure 3.12: Load distribution at the IP layer

Another aspect that can also be tributary to such behavior is the extent to which the algorithms are exploiting the topological flexibility of multi-layer networks. After the end of each

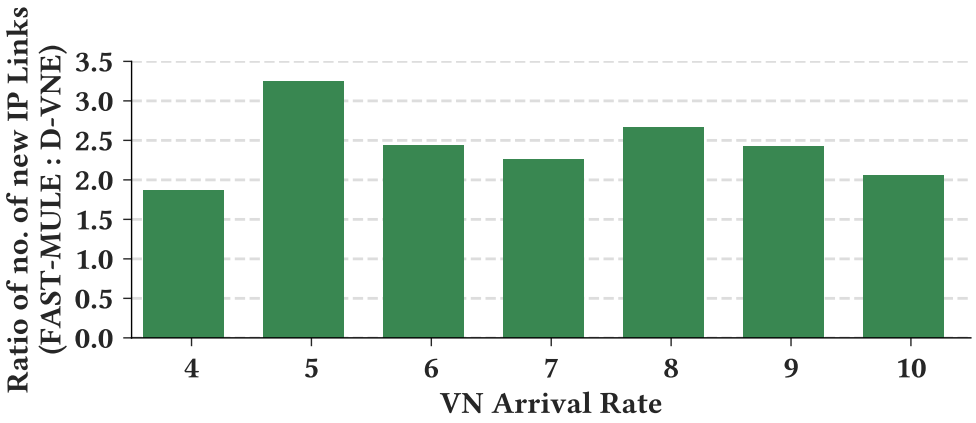
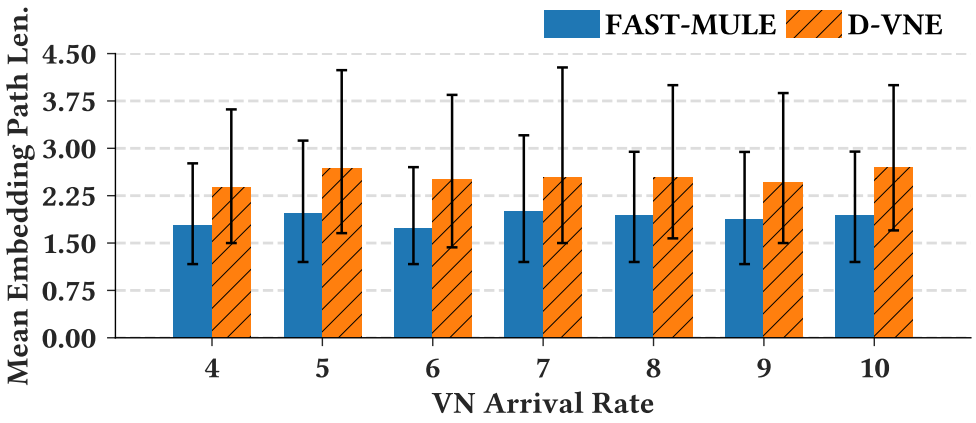


Figure 3.13: Ratio of newly created IP links (FAST-MULE : D-VNE) with varying load

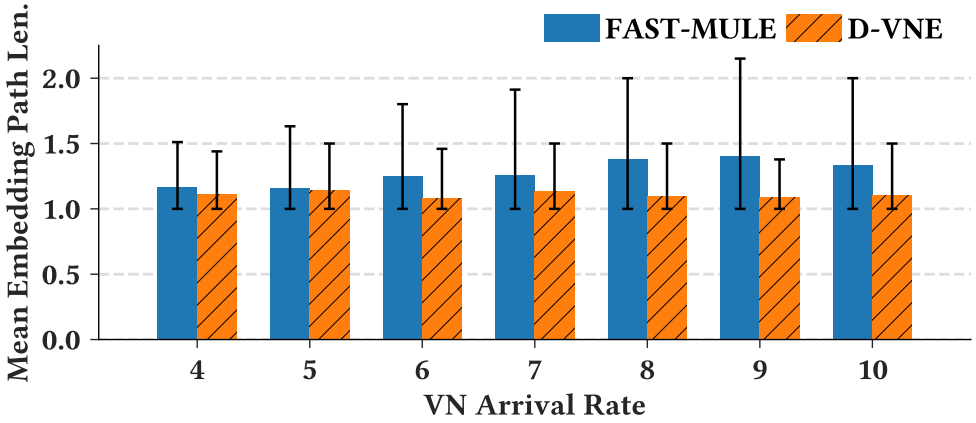
simulation we counted the total number of new IP links that were established by FAST-MULE and D-VNE, respectively, and present the ratio of these numbers in Figure 3.13. As we can observe, FAST-MULE consistantly created more IP links compared to D-VNE and hence was able to accept more VNs in the long run. Because of the higher number of IP links, the graph diameter reduced and resulted in shorter embedding paths, resulting in lower utilization of individual links. Because of the joint optimization approach, FAST-MULE was able to make better decisions regarding creation of new IP links and also for embedding paths, hence, the higher acceptance ratio and lower link utilization.

Topological Properties of the solution

For each simulation setting, we computed the mean embedding path length for both the virtual links and the newly created IP links and present the result in Figure 3.14. We observe from Figure 3.14(a) that FAST-MULE embedded the virtual links on shorter paths ($\approx$30%) compared to D-VNE. This is a combined effect of being able to create more IP links on the long run as well as the joint embedding of virtual nodes and virtual links whenever possible. We also present the mean embedding path length for the newly created IP links in Figure 3.14(b). Since the OTN is static and FAST-MULE established significantly more new IP links compared to that of D-VNE, IP link embedding paths became longer in case of FAST-MULE.



(a) VN to IP mapping



(b) IP to OTN mapping

Figure 3.14: Mean embedding path length

3.6 Related Works

Multi-Layer Embedding

A few works in the research literature addressed the problem of embedding in multi-layer networks [168, 169, 2]. In [168], the authors consider the problem of application-aware traffic embedding on IP/Multi-layer Protocol Switching(MPLS)- over-Optical Network. Traffic requests

are given between pairs of routers in the network, where each request has differentiated service requirements in terms of bandwidth, tolerable end-to-end delay, and tolerable end-to-end path availability. Here, the possibility of establishing new IP links is also considered when the IP/MPLS layer does not have sufficient capacity to meet the demand of a given request. However, in [168], the end points of the requests are fixed and it only addresses the link routing problem. Moreover, [168] neither does propose an optimal solution to the problem, nor presents complexity analysis of the proposed heuristic. Furthermore, the creation of new IP links is restricted to the pair of IP nodes that are not already connected.

In [169], the problem of Service Function Chaining (SFC) in IP-over-OTN networks is addressed. This work considers service function chains distributed across multiple data centers, where the data centers, representing the electronic layer, are interconnected via an optical network. The authors propose an algorithm in [169] to route each SFC request across the data centers. Therefore, even though the substrate network is a multi-layer one, the routing problem is addressed only in for optical layer. Furthermore, the placement of the network functions in the requested SFC is considered given, and the different segments (pair of service functions) that make up the SFC are routed sequentially.

To the best of our knowledge, the only work that considered the problem of multi-layer virtual network embedding while considering both node and link embedding is presented in [2]. Zhang *et al.*, proposed a heuristic for solving the multi-layer VNE problem for IP-over-DWDM networks. They also consider the possibility of modifying IP layer topology by allocating wavelengths from the underlying DWDM network. Zhang *et al.*, proposed a two step embedding process that first embeds the virtual nodes then the virtual links, which limits the solution space and hence the optimality of the embedding. In contrast, we propose an ILP formulation for optimally solving the multi-layer VNE problem. Also, our heuristic does not embed the virtual nodes and links independently from each other, rather tries to embed them simultaneously.

Multi-Layer Network Optimization

An orthogonal but related area of research in multi-layer network optimization focuses on the issue of capacity planning in multi-layer networks [170, 171]. During the initial capacity planning a traffic matrix for the IP layer is given and sufficient capacity needs to be allocated in both IP and Optical layers to support that traffic matrix. Different variants of the problem exist that take different technological constraints and deployment models into account [172, 173, 174, 175]. While the results from [172] are applicable for generic multi-layer network, however, that from [173] is specific to IP/MPLS-over-OTN-over-DWDM optical networks, with particular emphasis on the technological constraints of the OTN layer. Similar to [173], the work presented in [174] considers the capacity planning problem for OTN-over-DWDM networks.

Another research direction, that has been well explored in the research literature is that of protection planning for multi-layer networks [176, 177, 178, 179, 180]. Multi-layer protection planning involves deciding which layer will be in charge of protecting what, and coordinating the protection schemes across different layers [176, 181]. For instance, in [181], the authors showcase the limitations of traditional capacity planning in IP-over-OTN networks where each layer is treated in isolation. Subsequently, the authors motivate the advantages of coordinating across the different layers, and illustrate these benefits in Multi-Layer Restoration (MLR) planning. To achieve their goals, the authors compared MLR against restoration planning performed at the IP-layer alone and showed significant savings in the number of interfaces used and provisioned network resources (*i.e.*, wavelengths in their case). In [177], the authors address the problem of designing a multi-layer protection scheme for MPLS-over-OTN networks. They evaluate single and multi-layer survivability schemes under different spare capacity allocation strategies (*e.g.*, shared vs. dedicated). In the single layer survivability scheme, they propose to protect every Label Switched Path (LSP) against failures in the IP or in the OTN layer. Whereas, in the multi-layer survivability scheme, the OTN layer is protected against physical link and OXC failures, and the IP layer is protected against routers and IP/Optical interface failures.

In contrast to capacity and protection planning, in multi-layer VNE, the endpoint of the demands, *i.e.*, virtual node placement, is not known in advance, making this one a fundamentally different problem. Further, the body of research in multi-layer capacity and protection planning has demonstrated clear advantages of resource allocation when the layers are jointly optimized as opposed to considering them in isolation [182, 181, 177]. Our solution approach also takes a joint optimization approach to the multi-layer VNE problem.

3.7 Chapter Summary

In this chapter, we studied MULE, *i.e.*, multi-layer virtual network embedding on an IP-over-OTN substrate network. We proposed an ILP formulation, *OPT*-MULE, for optimally solving MULE and a heuristic, *FAST*-MULE, to address the computational complexity of the ILP. To the best of our knowledge, this is the first optimal solution to multi-layer VNE. Our evaluation of *FAST*-MULE shows that it performs within 1.47$\times$ of the optimal solution on average. *FAST*-MULE also outperformed state-of-the-art heuristic for multi-layer VNE and allocated $\approx$70% less resources on average while accepting $\approx$60% more VN requests on average. Finally, we also proved that our proposed heuristic computes the optimal solution for star shaped VNs with uniform bandwidth demand in polynomial time.

Chapter 4

Dedicated Protection for Survivable Virtual Network Embedding

4.1 Introduction

In this chapter, we study the problem of $1 + 1$-**Pro**tected **Vi**rtual **N**etwork **E**mbedding (1 + 1 − ProViNE) with the objective of minimizing resource provisioning cost in the Substrate Network (SN), while protecting each node and link in a Virtual Network (VN) request with dedicated backup resource in SN. The primary and backup embeddings need to be disjoint to ensure that a single physical node failure does not affect both the primary and the backup. If the primary embedding of a VN is affected by a physical node failure, the service provider operating on the VN should not incur a significant service disruption typical when migrating the whole or part of the VN to the backup. Indeed, during a single physical node failure, the disjoint primary and backup embeddings both accessible to the service provider enables the infrastructure provider to instantly switch traffic to the backup embedding without requiring any re-embedding decision. This capability of instantly switching traffic to the backup facilitates fast recovery within tens of milliseconds, which is a typical SLA between an OTN provider and customer [60, 61].

A major challenge in solving 1 + 1 − ProViNE is to find the primary and backup embedding at the same time. Relevant literature [3] shows that sequentially embedding the primary and backup can lead to failure in embedding even though a feasible embedding exists. In this regard, we propose **D**edicated P**ro**tection for Virtual **N**etwork **E**mbedding (DRONE), a suite of solutions for 1 + 1 − ProViNE. We focus on single node failure scenario since it is the most probable case [105, 106], and leave the multiple failure scenario for future investigation. Specifically, we make the following contributions in this chapter:

- **OPT-DRONE:** An Integer Linear Program (ILP) formulation for optimally solving 1 + 1 – ProViNE, improving on the quadratic formulation from previous work [3]. We also show that 1 + 1 – ProViNE is at least as hard as jointly solving balanced graph partitioning and minimum unsplittable flow problems, both of which are NP-Hard [107, 108].
- **FAST-DRONE**: A heuristic to tackle the computational complexity of OPT-DRONE and find solution to large problem instances in a reasonable time frame.
- Evaluation of the proposed solutions using realistic network topologies. Our key findings are: (i) FAST-DRONE uses ≈14.3% additional resources on average compared to OPT-DRONE, while executing 200× – 1200× faster; (ii) FAST-DRONE outperforms state-of-the-art solution for providing dedicated protection to VNs [3] by accepting 4× more VN requests on average.

The rest of the chapter is organized as follows. We begin with introducing the mathematical notations and a formal definition of 1 + 1 – ProViNE in Section 4.2. Then we present the ILP formulation of 1 + 1 – ProViNE, *i.e.*, OPT-DRONE in Section 4.3 followed by the details of FAST-DRONE in Section 4.4. Section 4.5 presents our evaluation of DRONE. Then we discuss he related works and contrast our solution with state-of-the-art in Section 4.6. Finally, we summerize the chapter contributions in Section 4.7.

4.2 1 + 1 - Protected Virtual Network Embedding Problem

In this section, we first present a mathematical representation of the inputs: the SN and the VN request. Then we formally define 1 + 1 – ProViNE.

4.2.1 Substrate Network

We represent the SN as an undirected graph, $G = (V, E)$, where V and E are the set of substrate nodes and links, respectively. The set of neighbors of each substrate node $u \in V$ is represented by $\mathcal{N}(u)$. Each substrate link $(u, v) \in E$ has the following attributes: (i) b_{uv} : bandwidth capacity of the link (u, v), and (ii) C_{uv} : cost of allocating unit bandwidth on (u, v) for provisioning a virtual link.

4.2.2 Virtual Network

We represent a VN as an undirected graph $\bar{G} = (\bar{V}, \bar{E})$, where $\bar{V}$ and $\bar{E}$ are the set of virtual nodes and virtual links, respectively. Each virtual link $(\bar{u}, \bar{v}) \in \bar{E}$ has bandwidth requirement $b_{\bar{u}\bar{v}}$. We also have a set of location constraints, $\mathcal{L} = \{L(\bar{u}) | L(\bar{u}) \subseteq V, \forall \bar{u} \in \bar{V}\}$, such that a virtual node $\bar{u} \in \bar{V}$ can only be provisioned on a substrate node $u \in L(\bar{u})$. The location constraint set for $\bar{u} \in \bar{V}$ contains all substrate nodes when there is no location constraint for $\bar{u}$. The binary variable $\ell_{\bar{u}u}$ represents the location constraint as follows:

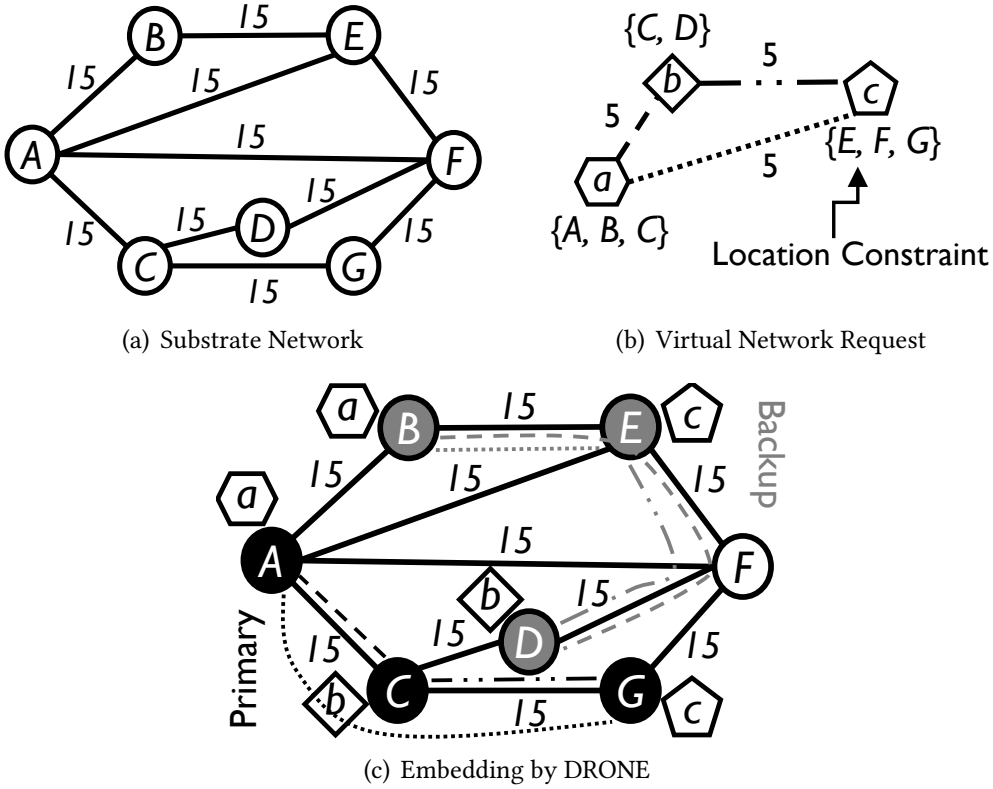
$$\ell_{\bar{u}u} = \begin{cases} 1 & \text{if } \bar{u} \in \bar{V} \text{ can be provisioned on } u \in V, \\ 0 & \text{otherwise.} \end{cases}$$

4.2.3 1 + 1 – ProViNE Problem Statement

Given a SN $G = (V, E)$, VN request $\bar{G} = (\bar{V}, \bar{E})$, and a set of location constraints $\mathcal{L}$, embed $\bar{G}$ on G such that:

- Each virtual node $\bar{u} \in \bar{G}$ has a primary and a backup embedding in the SN, satisfying the location constraint.
- For each virtual node $\bar{u} \in \bar{G}$, the substrate nodes used for the primary embedding are disjoint from the substrate nodes used for the backup embedding.
- Each virtual link $(\bar{u}, \bar{v}) \in \bar{E}$ has a primary and a backup embedding in the SN. A primary or backup embedding of a virtual link on the SN corresponds to a single path in the SN having at least $b_{\bar{u}\bar{v}}$ available bandwidth. The substrate paths corresponding to the primary and backup embedding of a virtual link $(\bar{u}, \bar{v}) \in \bar{E}$ are represented by $P_{\bar{u}\bar{v}}$ and $P'_{\bar{u}\bar{v}}$, respectively.
- Backup embedding of a virtual link is disjoint from the set of substrate paths used for primary embedding of the virtual links. The same disjointedness principle applies for the primary embedding.
- The total cost of provisioning bandwidth in the SN is minimum according to the following cost function:

$$\sum_{\forall(\bar{u},\bar{v})\in\bar{E}} \sum_{\forall(u,v)\in P_{\bar{u}\bar{v}}\cup P'_{\bar{u}\bar{v}}} C_{uv} \times b_{\bar{u}\bar{v}} \tag{4.1}$$



(a) Substrate Network

(b) Virtual Network Request

(c) Embedding by DRONE

Figure 4.1: Example VN embedding with DRONE

Therefore, a solution of 1 + 1 – ProViNE will yield two disjoint embeddings of a VN request on the SN while minimizing the given cost function (4.1). Figure 4.1 shows such an example with dark nodes and lines denoting the primary, and gray nodes and lines denoting the backup embedding of a VN request on a SN.

4.3 ILP Formulation: OPT-DRONE

1 + 1 – ProViNE's objective is to ensure fault tolerance of a VN by providing dedicated protection to each VN element with minimal resource overhead. This ensures that a *single physical element failure* does not bring down both the primary and backup embedding of the same VN element. To find an optimal solution, we first transform the input VN (Section 4.3.1), which ensures that the primary and the backup embedding are computed simultaneously, and then provide an ILP

Table 4.1: Summary of key notations

$G = (V, E)$	Physical Network
b_{uv}	Residual Bandwidth capacity of physical link $(u, v) \in E$
C_{uv}	Cost of allocating unit bandwidth on physical link $(u, v) \in E$ for provisioning a virtual link
$\bar{G} = (\bar{V}, \bar{E})$	Virtual Network Request
$b_{\bar{u}\bar{v}}$	Bandwidth requirement of virtual link $(\bar{u}, \bar{v}) \in \bar{E}$
$\mathcal{L}(\bar{u})$	Location constraint set for virtual node $\bar{u} \in \bar{V}$
$\ell_{\bar{u}u} \in \{0, 1\}$	$\ell_{\bar{u}u} = 1$ if $u \in \mathcal{L}(\bar{u})$, $u \in V, \bar{u} \in \bar{V}$
$\tilde{G} = (\tilde{V}, \tilde{E})$	Replica of virtual network request $\bar{G}$
$\hat{G} = (\hat{V}, \hat{E})$	Transformed virtual network request, $\hat{G} = \bar{G} \cup \tilde{G}$
$x_{uv}^{\hat{u}\hat{v}} \in \{0, 1\}$	$x_{uv}^{\hat{u}\hat{v}} = 1$ if $(u, v) \in E$ is on the embedded physical path for $(\hat{u}, \hat{v}) \in \bar{E}$
$y_{\hat{u}u} \in \{0, 1\}$	$y_{\hat{u}u} = 1$ if $\hat{u} \in \hat{V}$ is mapped to $u \in V$

formulation for the optimal embedding (Section 4.3.2). A glossary of key notations used in the ILP formulation is provided in Table 4.1.

4.3.1 Virtual Network Transformation

We formulate 1 + 1 – ProViNE as simultaneously embedding two copies of the same VN disjointly on the SN. To accomplish this goal, we first replicate the input VN, $\bar{G}$ to obtain a shadow VN, $\tilde{G} = (\tilde{V}, \tilde{E})$. $\tilde{G}$ has the same number of nodes and links as $\bar{G}$ and each shadow virtual link $(\tilde{u}, \tilde{v}) \in \tilde{E}$ has the same bandwidth requirement as the original virtual link $(\bar{u}, \bar{v}) \in \bar{E}$. We enumerate the nodes in the shadow VN $\tilde{G}$ by using the following transformation function: $\tau(\bar{u}) = \tilde{u}$.

Our transformed input now contains the graph $\hat{G} = (\hat{V}, \hat{E})$, *s.t.* $\hat{V} = \bar{V} \cup \tilde{V}$ and $\hat{E} = \bar{E} \cup \tilde{E}$. We now embed $\hat{G}$ on G in such a way that any node $u \in \bar{V}$ and any node $\tilde{u} \in \tilde{V}$ are not provisioned on the same physical node. Similar constraints apply on the virtual links as well.

4.3.2 ILP Formulation

We begin by introducing the decision variables (Section 4.3.2). Then we present the constraints (Section 4.3.2) followed by the objective function (Section 4.3.2).

Decision Variables

A virtual link is mapped to a physical path. The following decision variable indicates the mapping between a virtual link and a physical link.

$$x_{uv}^{\hat{u}\hat{v}} = \begin{cases} 1 & \text{if } (\hat{u},\hat{v}) \in \hat{E} \text{ is mapped to } (u,v) \in E, \\ 0 & \text{otherwise.} \end{cases}$$

The following decision variable represents the virtual node mapping:

$$y_{\hat{u}u} = \begin{cases} 1 & \text{if } \hat{u} \in \hat{V} \text{ is mapped to } u \in V, \\ 0 & \text{otherwise.} \end{cases}$$

Constraints

Link Mapping Constraints: We ensure that every virtual link is mapped to a non-zero length physical path by constraint (4.2). It also ensures that no virtual link is left unmapped. Physical link resource constraint is expressed using constraint (4.3). Finally, constraint (4.4) makes sure that the in-flow and out-flow of each physical node is equal except at the nodes where the endpoints of a virtual link are mapped. Constraint (4.4) ensures a continuous path between the mapped endpoints of a virtual link [183].

$$\forall(\hat{u},\hat{v}) \in \hat{E} : \sum_{\forall(u,v)\in E} x_{uv}^{\hat{u}\hat{v}} \geq 1 \tag{4.2}$$

$$\forall(u,v) \in E : \sum_{\forall(\hat{u},\hat{v})\in \hat{E}} x_{uv}^{\hat{u}\hat{v}} \times b_{\hat{u}\hat{v}} \leq b_{uv} \tag{4.3}$$

$$\forall\hat{u},\hat{v} \in \hat{V}, \forall u \in V : \sum_{\forall v\in\mathcal{N}(u)} (x_{uv}^{\hat{u}\hat{v}} - x_{vu}^{\hat{u}\hat{v}}) = y_{\hat{u}u} - y_{\hat{v}u} \tag{4.4}$$

The binary nature of the virtual link mapping decision variable along with the flow constraint prevents any virtual link being mapped to more than one physical path. This restricts the link mapping to the *Multi-Commodity Unsplittable Flow Problem* [108].

Node Mapping Constraints: Equations (4.5) and (4.6) ensure that a virtual node is mapped to exactly one physical node according to the given location constraints, respectively. Then, equation (4.7) ensures that a physical node does not host more than one virtual node from the same virtual network request.

$$\forall \hat{u} \in \hat{V}, \forall u \in V : \sum_{\forall u \in V} y_{\hat{u}u} = 1 \tag{4.5}$$

$$\forall \hat{u} \in \hat{V}, \forall u \in V : y_{\hat{u}u} \leq \ell_{\hat{u}u} \tag{4.6}$$

$$\forall u \in V : \sum_{\hat{u} \in \hat{V}} y_{\hat{u}u} \leq 1 \tag{4.7}$$

The virtual node embedding follows from the virtual link embedding since we do not have any cost associated with virtual node embedding. Therefore, the problem of coordinated node and link embedding is at least as hard as the *Multi-commodity Unsplittable Flow Problem with Unknown Sources and Destinations.*

Disjointedness Constraints: We need to ensure that every virtual link in $\bar{G}$ and its corresponding virtual link in $\tilde{G}$ is embedded on node and link disjoint paths in SN. To ensure this disjointedness property, we first constrain the virtual links in $\bar{G}$ and $\tilde{G}$ to be mapped on disjoint set of physical links using equations (4.8) and (4.9).

$$\forall (u,v) \in E : \sum_{\forall (\tilde{u},\tilde{v}) \in \tilde{E}} x_{uv}^{\tilde{u}\tilde{v}} = 0 \textit{ if } x_{uv}^{\bar{u}\bar{v}} = 1, \forall (\bar{u},\bar{v}) \in \bar{E} \tag{4.8}$$

$$\forall (u,v) \in E : x_{uv}^{\bar{u}\bar{v}} = 0 \textit{ if } \sum_{\forall (\tilde{u},\tilde{v}) \in \tilde{E}} x_{uv}^{\tilde{u}\tilde{v}} > 0, \forall (\bar{u},\bar{v}) \in \bar{E} \tag{4.9}$$

Then we forbid the virtual link endpoints of the primary embedding to be intermediate nodes on the path of backup embedding and vice versa using equations (4.10) and (4.11).

$$\forall u \in V : y_{\bar{u}u} = 0, \textit{ if } \sum_{\forall (\tilde{u},\tilde{v}) \in \tilde{E}} \sum_{\forall v \in \mathcal{N}(u)} x_{uv}^{\tilde{u}\tilde{v}} > 0 \tag{4.10}$$

$$\forall u \in V : \sum_{\forall (\tilde{u},\tilde{v}) \in \tilde{E}} \sum_{\forall v \in \mathcal{N}(u)} x_{uv}^{\tilde{u}\tilde{v}} = 0, \textit{ if } y_{\bar{u}u} = 1 \tag{4.11}$$

We also ensure that the physical paths corresponding to the virtual links in $\bar{G}$ and $\tilde{G}$ do not share any intermediate nodes. This constraint is necessary to ensure that a physical failure

does not affect a primary resource and its corresponding backup resource at the same time.

$$\forall u \in V: \sum_{\forall(\tilde{u},\tilde{v})\in\tilde{E}} \sum_{\forall v\in\mathcal{N}(u)} x_{uv}^{\tilde{u}\tilde{v}} = 0, \; \textit{if} \sum_{\forall(\bar{u},\bar{v})\in\bar{E}} \sum_{\forall v\in\mathcal{N}(u)} x_{uv}^{\bar{u}\bar{v}} > 0 \tag{4.12}$$

$$\forall u \in V: \sum_{\forall(\bar{u},\bar{v})\in\bar{E}} \sum_{\forall v\in\mathcal{N}(u)} x_{uv}^{\bar{u}\bar{v}} = 0, \; \textit{if} \sum_{\forall(\tilde{u},\tilde{v})\in\tilde{E}} \sum_{\forall v\in\mathcal{N}(u)} x_{uv}^{\tilde{u}\tilde{v}} > 0 \tag{4.13}$$

Objective Function

Our objective is to minimize the cost of provisioning bandwidth on the physical links. Therefore, we have the following objective function:

$$\text{minimize} \left(\sum_{\forall(\hat{u},\hat{v})\in\hat{E}} \sum_{\forall(u,v)\in E} x_{uv}^{\hat{u}\hat{v}} \times C_{uv} \times b_{\hat{u}\hat{v}} \right)$$

4.3.3 Hardness of 1 + 1 – ProViNE

As discussed earlier in Section 4.3.2, the coordinated node and link mapping without the disjointedness constraints is at least as hard as solving the NP-Hard *Multi-commodity Unsplittable Flow Problem with Unknown Source and Destinations*. State-of-the art literature reveals that this problem is also hard to approximate even when the source and destination of the flows are known. Recent research works have found $(2 + \varepsilon)$ approximation algorithms for line and cycle graphs, respectively [184]. However, finding constant factor approximation algorithms for general graphs still remains open [185]. With the added mutual exclusion constraints, the embedding problem becomes at least as hard as partitioning the SN while minimizing the cost of multi-commodity unsplittable flow with unknown sources and destinations in each of the partition. Even an easier version of this problem, *balanced graph partitioning*, is NP-hard [186] and does not have a constant factor approximation algorithm [107, 186]. This makes it challenging to devise an constant factor approximation algorithm for 1 + 1 – ProViNE.

4.4 Heuristic Solution: FAST-DRONE

Given the NP-hard nature of the 1 + 1 – ProViNE problem, we resort to a heuristic, *i.e.*, FAST-DRONE, for finding solutions within a reasonable time frame. First, we restructure 1 + 1 –

ProViNE for the ease of designing a heuristic, while keeping the original problem intact in its meaning (Section 4.4.1). Then we present our heuristic algorithm in detail (Section 4.4.2, Section 4.4.3, Section 4.4.4, and Section 4.4.5) to solve the restructured problem. We also analyze the running time of FAST-DRONE (Section 4.4.6) and provide a guideline on parallel implementation of FAST-DRONE on a multi-core machine (Section 4.4.7).

4.4.1 Problem Restructuring

We reformulate 1 + 1 – ProViNE as a variant of graph partitioning problem as follows:

Given an SN $G = (V, E)$, a VN request $\bar{G} = (\bar{V}, \bar{E})$, and a set of location constraints, $\mathcal{L} = \{L(\bar{u}) | L(\bar{u}) \subseteq V, \forall \bar{u} \in \bar{V}\}$ (Section 4.2.2), 1 + 1 – ProViNE requires to partition the graph G into two disjoint partitions $\mathcal{P}$ and $\mathcal{Q}$ such that:

- $\forall \bar{u} \in \bar{V}$, $\mathcal{P}$ has at least one element from each $L(\bar{u})$.
- $\forall \bar{u} \in \bar{V}$, $\mathcal{Q}$ has at least one element from each $L(\bar{u})$.
- The sub-graph induced by the elements of each set $L(\bar{u})$ in $\mathcal{P}$ (and $\mathcal{Q}$) is connected.
- The sum of costs of embedding $\bar{G}$ on $\mathcal{P}$ and $\mathcal{Q}$ is minimum according to the given cost function (4.1).

The sets $\mathcal{P}$ and $\mathcal{Q}$ are disjoint partitions of G where the primary and backup resources for $\bar{G}$ can be provisioned without violating the disjointedness constraint of 1 + 1 – ProViNE. An optimal $\mathcal{P}$ and $\mathcal{Q}$ will minimize the total cost of primary and backup link embedding. Such optimal $\mathcal{P}, \mathcal{Q}$ will yield the optimal solution to 1 + 1 – ProViNE.

Graph partitioning, which is an NP-hard problem [186], can be reduced to the aforementioned partitioning problem by relaxing the location constraint, *i.e.,* setting each set $L(\bar{u})$, $\forall \bar{u} \in \bar{V}$, equal to V. Once we have the two partitions, embedding the virtual links inside one partition is at least as hard as solving the NP-Hard *Multi-commodity Unsplittable Flow* problem [108], since we are not allowed to embed a virtual link over multiple substrate paths. In the next section, we present our heuristic algorithm based on this reformulation.

4.4.2 Heuristic Algorithm

In order to find a solution to 1 + 1 – ProViNE we need to partition the SN *s.t.* the total cost of embedding the virtual links in the partitions are minimized (Section 4.4.1). Our heuristic starts

with a seed mapping set containing the primary and backup mapping of one virtual node and goes through the following three phases to partition the SN and embed the VN:

Node Mapping Phase: Use the seed mapping and location constraint set to find a primary and backup node embedding for the other virtual nodes. This phase yields a partial partitioning of the SN. This partial partition acts as a seed that we grow to a complete partition of the SN into two disjoint subgraphs.

Partitioning Phase: Once we have a seed primary and backup partition from the node mapping phase, we grow the seed partition to include the rest of the substrate nodes into either of the partitions. At the end of this phase, all of the substrate nodes are either assigned to the primary or to the backup partition.

Link Mapping Phase: In this phase, we have the virtual node mapping and the primary and backup partition of the SN as input. We embed the virtual links in these partitions separately by using the Constrained Shortest Path First algorithm.

We run this three phase algorithm for different initial seed node mapping and retain the solution with the minimum cost. To generate different seed node mappings we identify the virtual nodes that have the smallest location constraint set. We call these virtual nodes the *most constrained virtual nodes*. Such virtual nodes may lead to infeasible embedding if they are not embedded first, since they have the fewest options for embedding. For each of these most constrained virtual nodes $\bar{u}_c$, we take every pair of substrate nodes from $\bar{u}_c$'s location constraint set $L(\bar{u}_c)$ and consider that pair as a primary and backup node embedding for $\bar{u}_c$. In this way, we generate a number of seed node mappings and execute the above-described three phase algorithm. In the rest of this section, we describe the individual phases in detail.

4.4.3 Node Mapping Phase

The node mapping phase follows a greedy approach to map the virtual nodes to its' primary and backup substrate nodes, while satisfying the location constraint. In this phase, we map the virtual nodes one at a time and select them in the increasing order of their location constraint set size. The rationale for following this order is that a virtual node with fewer possible locations for mapping is more constrained. Mapping a less constrained virtual node first might lead to infeasible mapping of the more constrained virtual node(s).

Node mapping is performed by the *MapVNodes* procedure presented in Algorithm 2. We first initialize the primary and backup node mapping sets $nmap_p$ and $nmap_s$, respectively, with the provided $seed$. Then we take one virtual node at a time according to the aforementioned order (line 5) and iterate over its location constraint set to find the best substrate node for primary

Algorithm 2: FAST-DRONE: Node mapping phase

Input: G: substrate network; $\bar{G}$: virtual network request; $seed$: seed mapping

1 **function** MapVNodes(G, $\bar{G}$, $seed$)
2 $nmap_p(seed.node) \leftarrow seed.primary$, $nmap_s(seed.node) \leftarrow seed.backup$
3 $taken(seed.primary) \leftarrow$ **false**, $taken(seed.backup) \leftarrow$ **false**
4 $\mathcal{P} \leftarrow \phi$, $\mathcal{Q} \leftarrow \phi$
`/* Sequence` $\bar{\mathcal{V}}$ `represents virtual nodes sorted in decreasing order of location constraint set size */`
5 **foreach** $\bar{u} \in \bar{\mathcal{V}}$ **do**
6 $best \leftarrow$ NIL
7 **foreach** $c \in location(\bar{u})$ **do**
8 **if *not*** $taken(c)$ ***and*** *IsBetterAssignment*$(G, \mathcal{P}, \mathcal{Q}, c, best)$ **then** $best \leftarrow c$
9 **if** $best \neq$ *NIL* **then**
10 $taken(best) \leftarrow$ **true**, $nmap_p(\bar{u}) \leftarrow best$, $\mathcal{P} \leftarrow \mathcal{P} \cup \{best\}$
11 $best \leftarrow$ NIL
12 **foreach** $c \in location(\bar{u})$ **do**
13 **if *not*** $taken(c)$ ***and*** *IsBetterAssignment*$(G, \mathcal{Q}, \mathcal{P}, c, best)$ **then** $best \leftarrow c$
14 **if** $best \neq$ *NIL* **then**
15 $taken(best) \leftarrow$ **true**, $nmap_s(\bar{u}) \leftarrow best$, $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{best\}$
16 **return** $\{nmap_p, nmap_s\}$

mapping (line 7 – 10) . After finding a primary mapping, we determine the corresponding backup mapping (line 12 – 15). While considering a substrate node $u \in V$ as primary mapping of a virtual node, we try to determine if u is a better choice compared to $best_u$, the best choice of substrate node that we have seen so far considering the node mappings we already have. This is evaluated using the *IsBetterAssignment* procedure. This procedure is outlined in Algorithm 3 and performs the following tests in the order they are listed. We choose this order to minimize the chances of not finding a solution and to create a partition that yields a close to optimal embedding.

- **Infeasibility Test (line 2):** *Does adding u to the primary mapping makes the backup mapping impossible to be connected and vice versa?* If the answer is *yes*, then we do not consider u for primary node mapping of the virtual node. Otherwise, we perform the next test.
- **Compact Mapping Test (line 4 – 6):** *Does considering u instead of $best_u$ in the primary (or backup) mapping decreases the mean shortest path length among the nodes currently*

present in the primary (or backup) mapping set? If the answer is *yes* then u is considered to be better than $best_u$. Otherwise, we perform the next test.

- **Connectivity Contribution Test (line 7 – 14):** *Does u contribute more* connectivity *to the mapping set (primary or backup) compared to $best_u$?* If the answer is *yes*, then $best_u$ is updated with u. We measure *connectivity contribution* using the following:
  - Number of connected components decreased in the current mapping set if u is considered instead of $best_u$ in the mapping set.
  - Number of links incident from u to the current mapping set compared to $best_u$.

We iterate over all possible substrate nodes u in the location constraint set of a virtual node and find the best among them for the mapping. We do the same iteration and tests again (line 24 of Algorithm 2) to find a backup mapping for that virtual node. We repeat this procedure for all the virtual nodes and we finally obtain a primary and backup mapping of the virtual nodes, $nmap_p$ and $nmap_s$, respectively. This primary and backup mapping sets acts as seed primary and backup partitions ($\mathcal{P}_0$ and $\mathcal{Q}_0$, respectively) that we grow to full partitions in the Partitioning phase.

Algorithm 3: FAST-DRONE: Check for better node assignment

```
1  function IsBetterAssignment(G, P, Q, u, best_u)
2      if not IsFeasiblePartition(G, P, Q, u) then return false
3      if best_u = NIL then return true
4      sp_reduction_current ← MeanSPReduction(G, P, Q, u)
5      sp_reduction_best ← MeanSPReduction(G, P, Q, best_u)
6      if sp_reduction_current > sp_reduction_best then return true
7      else if sp_reduction_current = sp_reduction_best then
8          decrease_current ← ComponentsReduced(G, P, u)
9          decrease_best ← ComponentsReduced(G, P, best_u)
10         if decrease_current > decrease_best then return true
11         else if decrease_current = decrease_best then
12             cut_current ← NumCutEdges(G, P, u)
13             cut_best ← NumCutEdges(G, P, best_u)
14             if cut_current > cut_best then return true
15     return false
```

4.4.4 Partitioning Phase

Algorithm 4: FAST-DRONE: Partitioning phase

Input: G: substrate network; $nmap_p$: primary node mapping; $nmap_s$: backup node mapping

function PartitionGraph(G, $nmap_p$, $nmap_s$)
- $\mathcal{P} \leftarrow \mathcal{Q} \leftarrow \phi$
- **foreach** $n_p \in nmap_p$ **do** $\mathcal{P} \leftarrow \mathcal{P} \cup \{n_p\}$
- **foreach** $n_s \in nmap_s$ **do** $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{n_s\}$
- $taken \leftarrow$ Array of size $|V|$, initialized with **false**
- **foreach** $v \in V$ **do**
  - **if *not*** $taken(v)$ **then**
    - **if *not*** *IsFeasiblePartition*$(G, \mathcal{P}, \mathcal{Q}, v)$ **then** $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{v\}$
    - **else**
      - $x \leftarrow$ MeanSPReduction$(G, \mathcal{P}, \mathcal{Q}, u)$
      - $y \leftarrow$ MeanSPReduction$(G, \mathcal{Q}, \mathcal{P}, u)$
      - **if** $x > y$ **then** $\mathcal{P} \leftarrow \mathcal{P} \cup \{v\}$
      - **else if** $x = y$ **then**
        - $\delta_p \leftarrow$ ComponentsReduced$(G, \mathcal{P}, v)$
        - $\delta_s \leftarrow$ ComponentsReduced$(G, \mathcal{Q}, v)$
        - **if** $\delta_p > \delta_s$ **then** $\mathcal{P} \leftarrow \mathcal{P} \cup \{v\}$
        - **else if** $\delta_p = \delta_s$ **then**
          - $cut_p \leftarrow$ NumCutEdges$(G, \mathcal{P}, v)$
          - $cut_s \leftarrow$ NumCutEdges$(G, \mathcal{Q}, v)$
          - **if** $cut_p > cut_s$ **then** $\mathcal{P} \leftarrow \mathcal{P} \cup \{v\}$
          - **else if** $cut_p < cut_s$ **then** $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{v\}$
          - **else** Assign u to the smaller partition
        - **else** $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{v\}$
- **return** $\{\mathcal{P}, \mathcal{Q}\}$

Given two seed primary ($\mathcal{P}_0$) and backup ($\mathcal{Q}_0$) partitions obtained from the node mapping phase, we partition the substrate network G into two disjoint partitions $\mathcal{P}$ and $\mathcal{Q}$ for the primary and backup embeddings of the virtual network, respectively. The partitioning process is performed using the *PartitionGraph* procedure (Algorithm 5). We consider the substrate nodes that are not already assigned to any of the partitions one at a time, and perform the following tests in the order they are listed. Such order is chosen for similar reasons as discussed in the node mapping phase.

- **Infeasibility Test:** Does adding u to $\mathcal{P}$ makes the partition $\mathcal{Q}$ impossible to be connected (line 6)? If the answer is *yes*, then we do not consider u for $\mathcal{P}$, rather we add u to $\mathcal{Q}$.
- **Compact Partition Test:** Does including u to $\mathcal{P}$ reduce shortest path length more than that reduced when u is added to $\mathcal{Q}$ (9 – 11)? If the answer is *yes*, then add u to $\mathcal{P}$, otherwise evaluate the next test. `Mean-SP-Reduction` procedure computes the reduction in mean shortest path length within a partition if a candidate node is added to that partition.
- **Connectivity Contribution Test:** We determine whether a candidate node $u \in V$ contributes more connectivity to $\mathcal{P}$ or to $\mathcal{Q}$ by evaluating the following:
  - *Does including u in $\mathcal{P}$ reduces more the number of components compared to adding u to $\mathcal{Q}$ (line 14 – 16)?* If the answer is *yes*, then u is added to $\mathcal{P}$, otherwise we evaluate the next criterion. *Components-Reduced* procedure computes the reduction in number of components if a candidate node is added to a partition.
  - *Does the candidate node $u \in V$ has more substrate links going to $\mathcal{P}$ compared to $\mathcal{Q}$ (line 19 – 24)?* If the answer is *yes* then u is added to $\mathcal{P}$, otherwise u is added to $\mathcal{Q}$. *Num-Cut-Edges* procedure computes the number of substrate links from a candidate node to a partition.

  Load Balancing Test: If all the previous tests fail to assign a $u \in V$ to either $\mathcal{P}$ or $\mathcal{Q}$, then we assign u to $\mathcal{P}$ if $|\mathcal{P}| < |\mathcal{Q}|$, otherwise u is assigned to $\mathcal{Q}$.

PartitionGraph procedure iterates over all the unassigned substrate nodes $u \in V$ and assigns u to either $\mathcal{P}$ or to $\mathcal{Q}$. At the end of this phase, we have two disjoint partitions, each of them has at least one node from each of the location constraint sets. Therefore, this partitioning conforms to the conditions as described in Section 4.4.1.

4.4.5 Link Mapping Phase

Given the two disjoint partitions, $\mathcal{P}$ and $\mathcal{Q}$, and the node mappings for the virtual nodes in each partition, we use constrained shortest path first algorithm to map a virtual link to a substrate path inside a partition. Application of shortest path based algorithms are common practice in cases when virtual links cannot be split and embedded on multiple substrate paths [187]. We also have this constraint in 1 + 1 – ProViNE.

All of the three phases are combined and presented in the *FAST-DRONE* procedure (Algorithm 7). Line 2 corresponds to the node mapping phase, Line 3 represents the partition growing phase, and finally lines 4 and 5 give us the link mappings.

Algorithm 5: FAST-DRONE algorithm

Input: G: substrate network; $\bar{G}$: virtual network request; $location$: location constraints

function FAST-DRONE(G, $\bar{G}$, $location$)
  $\{nmap_p, nmap_s\} \leftarrow$ MapVNodes($G, \bar{G}, location$)
  $\{\mathcal{P}, \mathcal{Q}\} \leftarrow$ PartitionGraph($G, nmap_p, nmap_s$)
  $emap_p \leftarrow$ EmbedAllVLinks($G, \bar{G}, \mathcal{P}, nmap_p$)
  $emap_s \leftarrow$ EmbedAllVLinks($G, \bar{G}, \mathcal{Q}, nmap_s$)
  Compute $embedding\_cost$ from $emap_p$ and $emap_s$
  return $\{nmap_p, emap_p, nmap_s, emap_s, cost\}$

4.4.6 Running Time Analysis

Before going to the analysis we first introduce the following notations:

- n = Number of vertices in the substrate network
- n' = Number of vertices in the virtual network
- m = Number of edges in the substrate network
- m' = Number of edges in the virtual network
- σ = Maximum size of a location constraints set for any virtual node
- δ = Maximum degree of a substrate node

We analyze the running time of `FAST-DRONE` procedure by analyzing the running time for each of the phases as follows:

Node Mapping Phase: Sorting the virtual nodes requires $O(n' \log n')$ time. Then for each of these n' virtual nodes, we traverse its location constraint set, which can have $\leq \sigma$ elements. For each of these $O(\sigma)$ nodes, we perform: (i) feasibility check (ii) compute the reduction in shortest path length (iii) compute the decrease in number of components and (iv) compute the number of edges incident form the candidate substrate node to the current set of mappings.

Task (i) can be accomplished in $O(n+m)$ time by simply keeping a disjoint set data structure with $O(n)$ elements, and perform union operation on the data structure. Task (ii) can take up to $O(n'^3)$ time. Task (iii) can be performed in $O(n + m)$ time in the worst case using a disjoint

set data structure. Finally for task (iv), the number of edges incident from a candidate substrate node to a mapping set can be computed in $O(\delta)$ time. Therefore, the mapping phase runs in $O(n'\sigma(n + m + \delta + n'^3))$ time.

Graph Partitioning Phase We iterate over $O(n)$ unassigned substrate nodes and perform similar steps as in the node mapping phase. Therefore, the time complexity of each iteration is the same as the four tasks described for the node mapping phase. Hence, partitioning the graph requires $O(n(n + m + \delta + n'^3))$ time.

Link Mapping Phase For the link mapping phase, we compute shortest path between the mapped nodes for each of the m' virtual links using Dijkstra's shortest path algorithm. This requires $O(m'm \log n)$ time in total.

Overall, the running time of the proposed heuristic is: $O((n'\sigma + n)(n + m + \delta + n'^3) + m'm \log n)$.

4.4.7 Parallel Implementation of FAST-DRONE

The proposed heuristic, *i.e.*, FAST-DRONE can be implemented as a multi-threaded program to utilize multiple CPU cores. We observed in Algorithm 2 that embedding of the first virtual node to its primary and backup substrate nodes has the most impact on the subsequent embeddings. To mitigate this impact, we can consider all possible initial primary/backup embedding combinations for the most constrained virtual node, *i.e.*, the virtual node with the smallest location constraint set and then run the rest of the heuristic. We can parallelize this process by executing each run of the heuristic with one combination of primary/backup embedding of the first virtual node on a separate thread running on one CPU core. After the parallel executions finish, we can choose the best embedding among all the parallel executions.

4.5 Performance Evaluation

We evaluate the proposed solutions for 1 + 1 – ProViNE through extensive simulations. We perform simulations using both randomly generated network topologies with various connectivity levels and a real ISP topology. Section 4.5.1 describes the simulation setup in detail and Section 4.5.2 defines the performance metrics. Then we present our evaluation results focusing on the following two aspects.

First, we perform micro-benchmarking of our solutions and compare with PAR [3], a recent work on survivable virtual infrastructure embedding with dedicated resources. For the micro-benchmarking scenario, we consider each VN embedding request in isolation and assume that

the VN can always be embedded on the SN. Under these assumptions, we measure the resource efficiency of our proposed solutions and compare our results with that of [3].

Second, we perform steady state analysis of FAST-DRONE and compare with that of PAR [3]. The steady state analysis considers VN arrivals and departures over a period of time and also considers the possibility of failing to embed a VN request on the SN. The steady state analysis provides valuable insight on the number of accepted VNs and the substrate resource utilization in a longer time frame.

4.5.1 Simulation Setup

Testbed

We have implemented OPT-DRONE, the ILP based optimal solution for 1 + 1 – ProViNE using IBM ILOG CPLEX 12.5 C++ libraries. The heuristic is also implemented in C++. We implemented the heuristic as a multi-threaded program by following the guidelines in Section 4.4.7. Both the heuristic and CPLEX solutions were run on a machine with hyper-threaded 8×10 core Intel Xeon E7-8870 CPU and 1 TB of memory. Both the CPLEX and heuristic implementations spawn up to 160 threads to saturate all the processing cores during their executions. We have developed an in-house discrete event simulator that simulates the arrival and departure of VNs for the steady state scenario.

Substrate Network Topology

We have generated random topologies for the micro-benchmarking scenario by varying the number of substrate nodes from 50 to 200 in increments of 25, and varying the Link-to-Node Ratio (LNR) from 1.2 to 2.2 in steps of 0.1. We used SNs with different LNR to study the impact of substrate network's connectivity on FAST-DRONE's performance. For the steady state scenario, we have used the topology of a large ISP (AS-6461) from the Rocketfuel ISP topology dataset [164] containing 141 nodes and 374 links. We used random integers uniformly distributed between 35000 Mbps and 40000 Mbps as link bandwidth, since link bandwidth is not specified in [164].

Virtual Network Topology

We generated three types of VN topologies for the micro-benchmarking scenario: ring, star and randomly connected graphs with up to 16 nodes to study the impact of different types of VNs

on FAST-DRONE's performance. As stated earlier, the micro-benchmarking scenario evaluates the resource efficiency of the algorithms given that the substrate network's capacity and the virtual nodes' location constraints yield a feasible embedding. In order to ensure that the VNs have a feasible embedding, we have iteratively grown the VNs from an input SN as follows.

We first start with an empty VN and add exactly one virtual node and some virtual links to the VN in each iteration. During an iteration, we first randomly select a substrate node and its neighbor as the primary and backup embedding for the new virtual node being added. Then we find paths in the SN from these primary and backup embeddings to the primary and backup embeddings of the existing virtual nodes, respectively. These existing virtual nodes are selected according to the type of VN (*e.g.*, ring, star, random) that we are trying to grow. Each pair of these newly found primary and backup paths in the SN, correspond to a virtual link between the virtual node being added and the selected virtual nodes from the already grown VN. While computing the paths, we maintain the disjointedness invariant of 1 + 1 – ProViNE as described in Section 4.2.3. This procedure ensures that the grown VN has at least one valid embedding on the SN. This process is continued until a VN of a desired size is found.

For the steady state scenario, we generated VNs with random connectivity. We set the VN connectivity level at 50% and vary the number of virtual nodes from 4 to 8. The virtual link bandwidth requirements are integers chosen uniformly between 12000 Mbps and 15000 Mbps. The VN arrival rate follows a Poisson distribution with a mean from 4 to 10 VNs per 100 time units. The VN life time is exponentially distributed with a mean of 1000 time units. These parameters have been chosen in accordance with the standards used in the related literature [55, 188]. For the location constraint of a virtual node, we randomly choose a substrate node and all the nodes reachable within a 3-hop radius.

4.5.2 Performance Metrics

- **Embedding Cost:** The embedding cost is the cost of provisioning bandwidth for the virtual links and their backups, computed using (4.1).
- **Execution Time:** The time required for an algorithm to find the solution to 1 + 1 – ProViNE.
- **Mean embedding path length:** For a given VN request, this is the mean of the substrate path lengths corresponding to the virtual link embeddings.
- **Acceptance Ratio:** Acceptance ratio is the fraction of VN requests that have been successfully embedded on the SN over all the VN requests.

- **Utilization:** We compute the utilization of a substrate link as the ratio of total bandwidth allocated to the embedded virtual links to that substrate link's capacity.

4.5.3 Micro-benchmarking Results

Our micro-benchmarking evaluation scenario focuses on the following aspects: (i) comparing the resource efficiency of the proposed heuristic (FAST-DRONE) with that of the optimal (OPT-DRONE); (ii) analyzing the impact of VN topology type; (iii) analyzing the impact of substrate network connectivity levels; (iv) demonstrating the scalability of FAST-DRONE and (v) comparing DRONE with PAR [3].

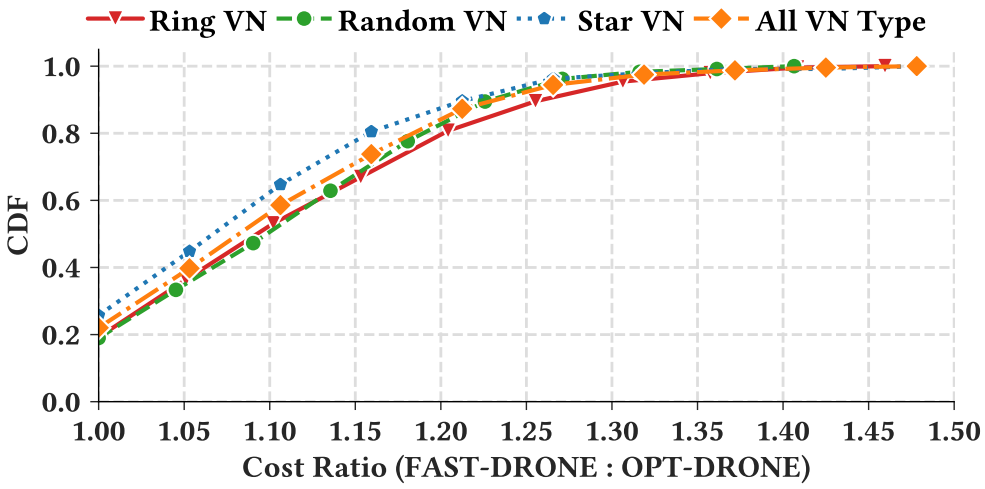


Figure 4.2: Comparison between OPT-DRONE and FAST-DRONE

Comparison between OPT-DRONE and FAST-DRONE

In this section, we present the results on how much extra resource is provisioned by FAST-DRONE compared to OPT-DRONE. This extra resource usage is measured as the ratio of FAST-DRONE's cost to OPT-DRONE's cost since our cost function is proportional to the total bandwidth allocated for the VN. Figure 4.2 shows the Cumulative Distribution Function (CDF) of cost ratio for different types of VN requests as well as the CDF for all types combined. A point (x, y) on this curve gives us the fraction y of total VN requests with cost ratio $\leq x$. This plot

shows that about 70% VN requests are embedded by FAST-DRONE with at most 15% extra resources, while 90% VN requests are embedded with at most 23% extra resources compared to OPT-DRONE. On average this extra resource provisioning is 14.3% over all VN request types.

Impact of VN Request Type

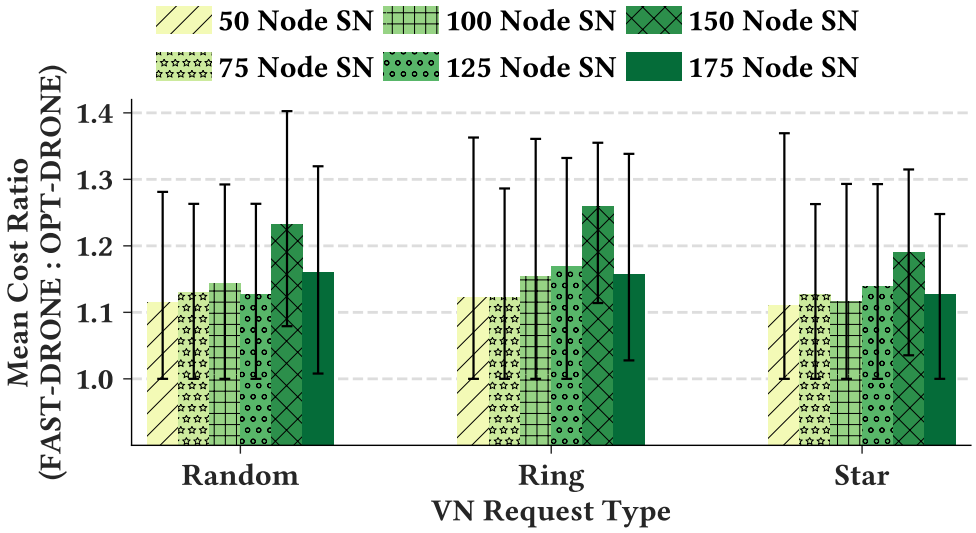
Figure 4.3(a) presents result for the cost ratio of different types of VN requests on different sizes of SNs. A take away from this result is that FAST-DRONE performs better for star VN topologies compared to ring and randomly connected VN topologies. The reason behind such behavior is that only the center node in a star topology imposes high disjointedness requirement. On the other hand, all nodes in a ring or a randomly connected VN topology impose similar disjointedness requirement. This intensifies resource contention while allocating disjoint paths in the SN leading to longer paths, hence, the higher cost ratio.

We also compute the mean substrate path lengths for the embedded virtual links to validate this finding and present the result along with 5th and 95th percentile error bars in Figure 4.3(b). The difference between mean embedded path length obtained by FAST-DRONE compared to OPT-DRONE is slightly higher for ring and randomly connected VN topologies (15% and 13%, respectively) compared to star VN topologies (12.5%). If we consider the 95th percentile of the spectrum, this difference is more significant, *i.e.*, 13.8%, 12%, and 10%, for ring, random, and star VN topologies, respectively.

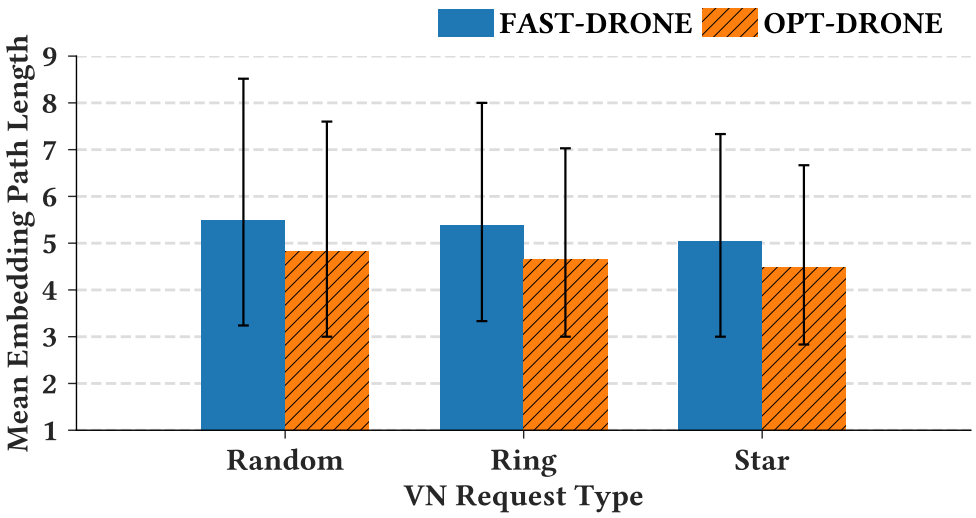
Impact of Substrate Network Connectivity

In this section, we present results on how the connectivity level of the underlying SN impacts the performance of FAST-DRONE. We varied the LNR of the generated substrate networks from 1.2 to 2.2 in increments of 0.1 and measured the mean FAST-DRONE to OPT-DRONE cost ratio for each case. Figure 4.4(a) shows the mean cost ratio with 5th and 95th percentile error bars against different LNRs. This plot gives us an idea about a good operating region for FAST-DRONE. As we can observe, FAST-DRONE allocates about 15% extra resources compared to the optimal solution for SNs having an LNR ≤ 1.8. For higher LNR, the increased path diversity may lead to more sub-optimal solution since the heuristic does not explore all the paths to keep the running time fast.

We also compute the mean of embedded substrate path lengths corresponding to the virtual links and present the results in Figure 4.4(b). The takeaway from this figure is that, a lower LNR in the SN, *i.e.*, sparse SNs cause both OPT-DRONE and FAST-DRONE to select longer paths for virtual link embedding. This is due to less path diversity in the SN. However, with increasing



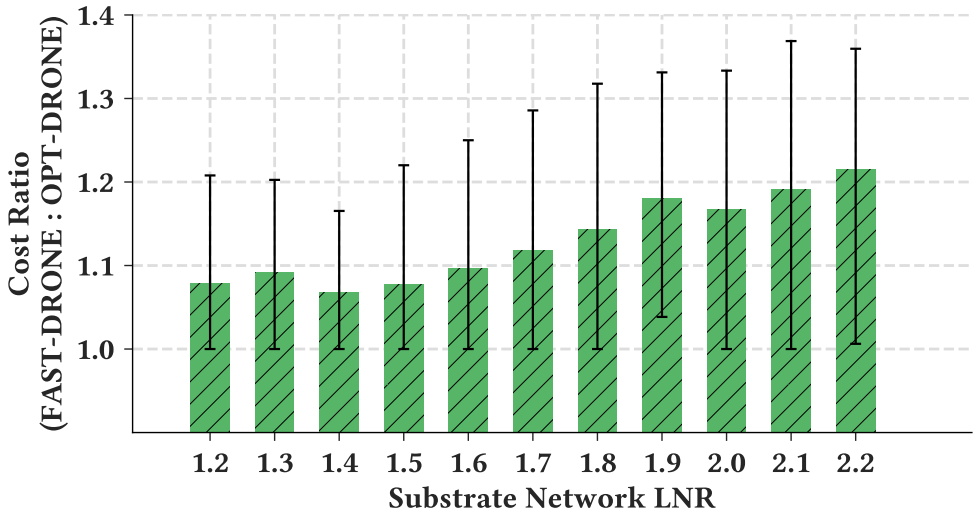
(a) Impact on cost ratio



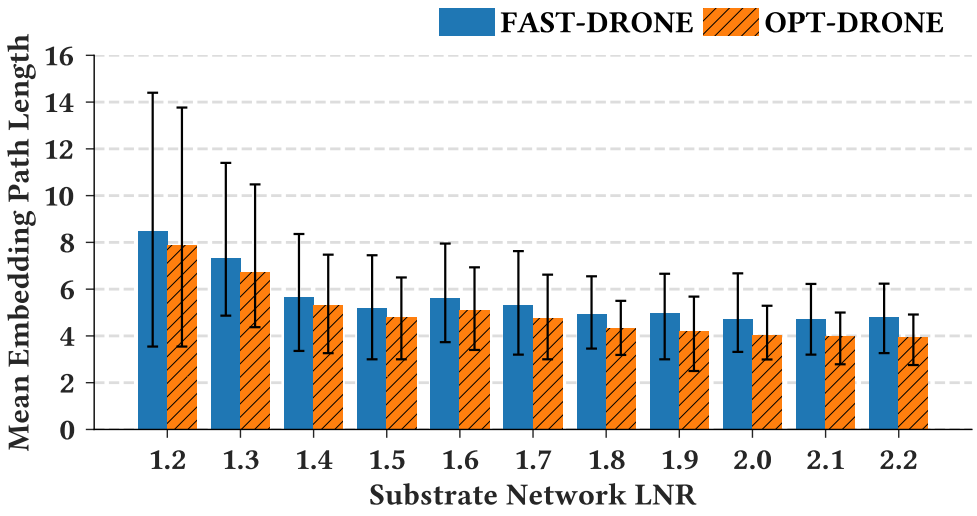
(b) Impact on mean embedding path length

Figure 4.3: Impact of VN request type

LNR, more paths become available in the SN and the mean path lengths for embedding virtual links become shorter. However, in line with the previous observation, FAST-DRONE explores



(a) Impact on cost ratio



(b) Impact on mean embedded path length

Figure 4.4: Impact of SN connectivity

a much smaller set of paths to keep the running time fast. As a result, the gap between mean embedded path lengths for OPT-DRONE and FAST-DRONE increases.

Scalability of Heuristic

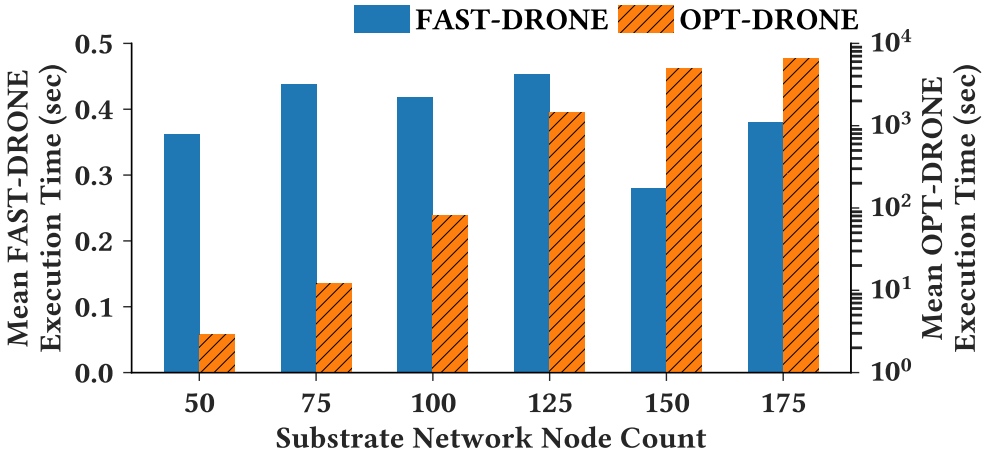
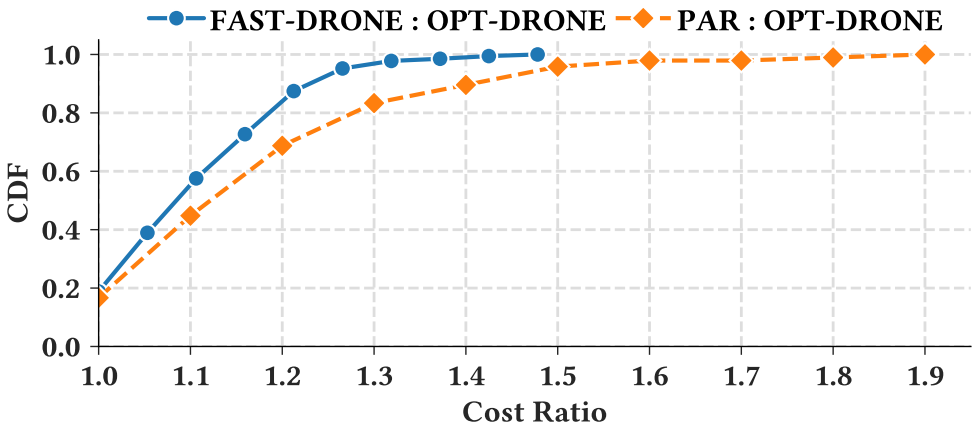


Figure 4.5: Comparison of execution time

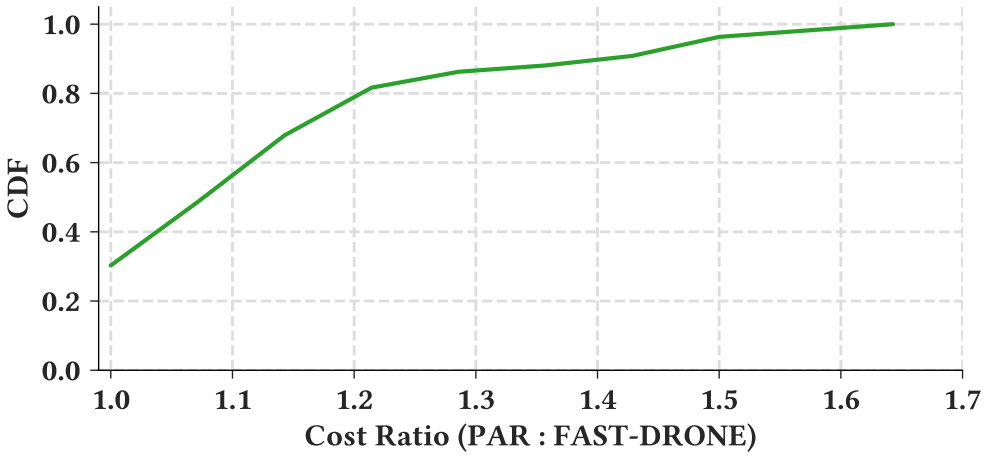
To demonstrate the scalability of FAST-DRONE we show the execution times of FAST-DRONE and OPT-DRONE on same problem instances in Figure 4.5. As it turns out, FAST-DRONE takes less than 500 ms on average over all test cases, whereas OPT-DRONE takes more than 2 minutes to run on average on the smallest SN instance. For larger instances, the execution time exponentially increases for OPT-DRONE and becomes in the order of hours (*e.g.*, about 110 minutes on average for a 175 node SN). We found FAST-DRONE to be 200 – 1200 times faster than OPT-DRONE depending on the problem instance. With our current setup OPT-DRONE did not scale beyond SNs with more than 175 nodes.

Comparison with PAR [3]

PAR [3] is a greedy heuristic for embedding a VN request on a SN with $1 + 1$-protection. PAR maximizes the probability of accepting a VN request by first embedding the virtual nodes on substrate nodes with higher residual node capacities. After node embedding, PAR embeds the virtual links using a modified version of Suurballe's algorithm [189]. In our case, we do not have node capacities. Therefore, we first implemented PAR by randomly mapping a virtual node $\bar{u} \in \bar{V}$ to a substrate node within its location constraint set $L(\bar{u})$. However, such random mapping lead to infeasible solutions almost all the time. Then we used our proposed MapVNodes (Algorithm 2) procedure to map the virtual nodes. The link embedding was done exactly



(a) Comparison with OPT-DRONE



(b) Cost ratio of PAR to FAST-DRONE

Figure 4.6: Comparison between FAST-DRONE and PAR [3]

the same way as described in [3]. It is worth noting that even after the modification in the node embedding, PAR could only find solutions for ≈12% test cases in our simulation setting.

We first compare how much extra resource is allocated by PAR and FAST-DRONE compared to the optimal solution (OPT-DRONE). For this comparison, we compute the ratio of costs (cost is computed using (4.1)) between PAR and OPT-DRONE, and FAST-DRONE and OPT-DRONE. Figure 4.6(a) shows the CDF of these cost ratios. This plot shows that in 90% cases,

PAR allocates up to 40% additional resources compared to the optimal, whereas, FAST-DRONE allocates up to 23% additional resources. On average, the amount of extra resource allocated compared to the optimal is 25% and 14.3% for PAR and FAST-DRONE, respectively. We also compute the ratio of PAR's cost to that of FAST-DRONE and plot the CDF in Figure 4.6(b) to see how much FAST-DRONE improves over PAR. This plot shows that PAR never performs better than FAST-DRONE and allocates up to 40% extra resources compared to FAST-DRONE at the 90th percentile. On average, we found PAR to allocate 17.5% additional resources compared to FAST-DRONE.

4.5.4 Steady State Analysis

Our steady state analysis focuses on the following aspects: (i) comparing the acceptance ratio obtained by FAST-DRONE with that of PAR [3] under different loads, *i.e.*, VN arrival rates (Section 4.5.4), (ii) compare the impact of FAST-DRONE on the load distribution on substrate links with that of PAR (Section 4.5.4), and (iii) analyze topological properties of the solutions (Section 4.5.4). We perform the steady state analysis using the VN arrival rate and duration parameters described in Section 4.5.1 for a total of 10000 time units. The total number of VNs used in this simulation was varied between 400 to 960.

Acceptance Ratio

In this section, we report our findings on the acceptance ratio obtained by FAST-DRONE and compare that with the acceptance ratio obtained by using PAR [3]. We consider the first 1000 time units of the simulation as the warm up period and discard the values from this duration. We take the mean of the acceptance ratio obtained during the rest of the simulation and report it along with 5th and 95th percentile values under varying VN arrival rates in Figure 4.7. We can see from the results that FAST-DRONE outperforms PAR in all cases and accepts $3.8\times$ more VNs on average over all VN arrival rates. There are two possible reasons contributing this difference: (i) either the network is being quickly saturated by one algorithm leading to its lower acceptance ratio, or (ii) the one algorithm having lower acceptance ratio is not sufficiently exploring the search space to be able to utilize the SN resources, hence, leaving SN resources unused. In the next section, we analyze how FAST-DRONE and PAR distribute the load on the SN to gain more insight into the difference between their achieved acceptance ratios.

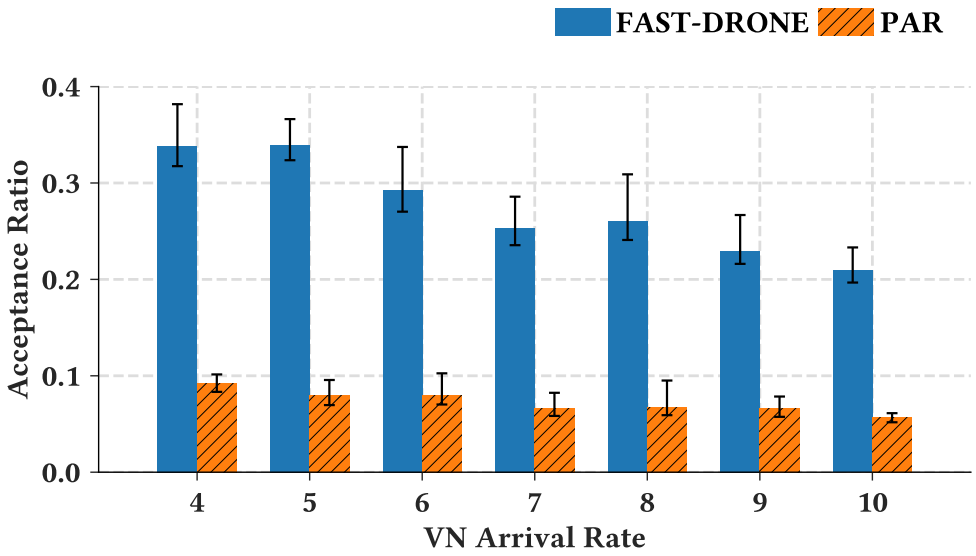


Figure 4.7: VN acceptance ratio

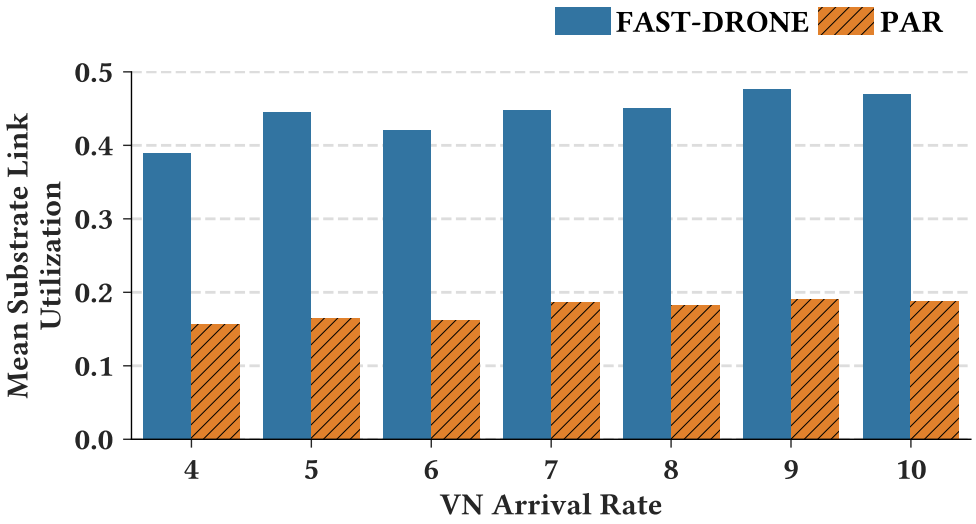
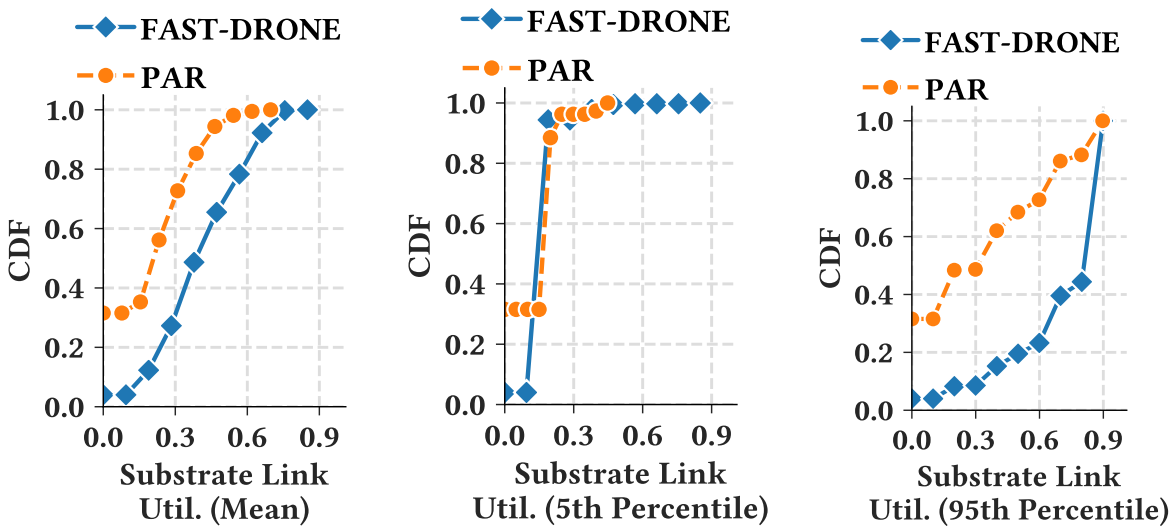


Figure 4.8: Mean substrate link utilization with varying load

Load Distribution on SN

We measure the utilization of each substrate link at each VN arrival and departure events, and compute the mean utilization for each substrate link. We first present results showing

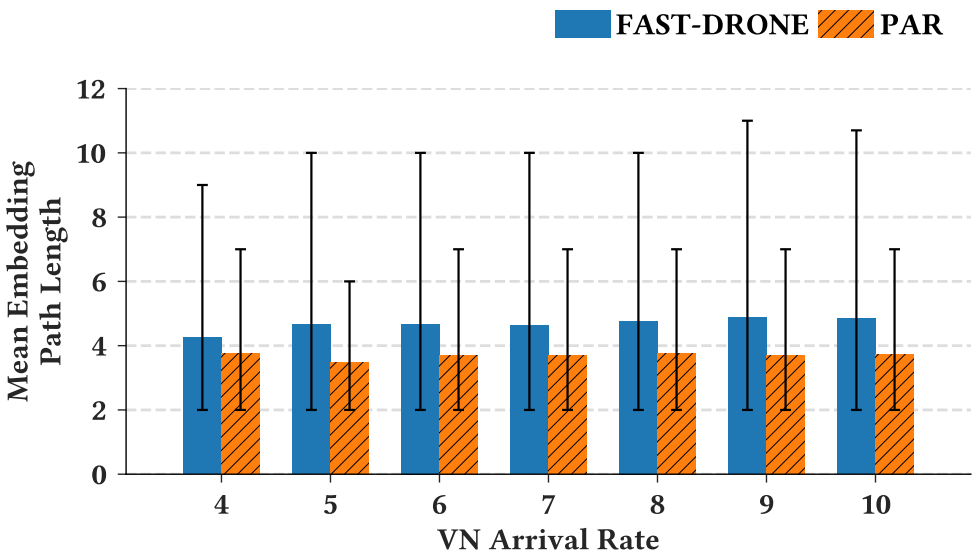


(a) Link utilization (Mean) (b) Link utilization (5th percentile) (c) Link utilization (95th percentile)

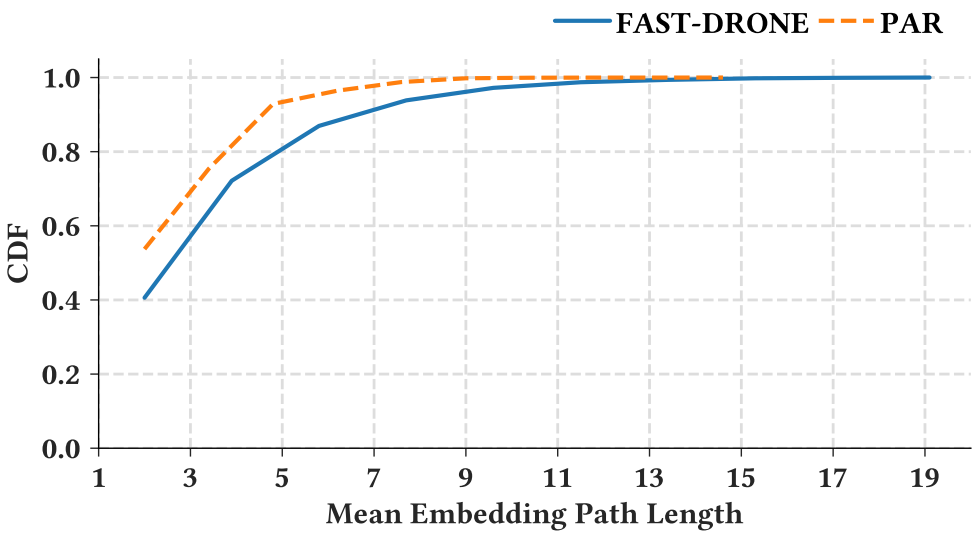
Figure 4.9: Load distribution on substrate network

the mean substrate link utilization with varying VN arrival rates in Figure 4.8. As we can see, there is a slight increase in the mean substrate link utilization with increasing VN arrival rate. Also substrate link utilization is on average 2.5× higher for FAST-DRONE compared to PAR. However, this plot, representing the mean utilization, fails to capture the variance in link utilization and does not give us much insight into how the load is distributed across the SN.

In order to capture the essence of load distribution across the substrate links, we compute the CDF of average, 5th and 95th percentile substrate link utilization for each VN arrival rate. However, we found the CDFs for different VN arrival rates to follow similar trend, hence, we combined the CDFs for different VN arrival rates into one and present the results in Figure 4.9. It is evident from Figure 4.9 that a significant portion of the substrate links (≈30%) remain unused by PAR throughout the simulation. In case of FAST-DRONE, the fraction of unused substrate links is less than 5%. In addition, for any level of utilization x, the fraction of substrate links having utilization $\geq x$ is larger for FAST-DRONE compared to PAR for all three cases, *i.e.*, average, 5th and 95th percentile. When load distribution is combined with acceptance ratio, we observe that despite having unused capacity in the SN, PAR yields lesser acceptance ratio compared to FAST-DRONE. This indicates that FAST-DRONE is exploring larger portion of the solution space compared to PAR, hence, the increased acceptance ratio.



(a) Mean embedding path length vs. VN arrival rate



(b) CDF of mean embedded path length

Figure 4.10: Topological properties of solutions

Topological Properties of the Solutions

We compute the mean embedding path lengths for the virtual links and present the results in Figure 4.10. Figure 4.10(a) shows the variation in mean path length with varying VN arrival rate, and Figure 4.10(b) shows the CDF of mean path lengths over all VN arrival rates. The results show that for similar VN arrival rate and also for all VN arrival rates FAST-DRONE yields embeddings that have slightly longer mean embedding path length compared to PAR. According to our cost function (4.1), longer embedding paths result into an increased cost. Therefore, results from these plots are counterintuitive when they are compared to that from Section 4.5.3, which indicated that FAST-DRONE yields more resource efficient embeddings compared to PAR. The reason behind this slightly longer paths during the steady state scenario is that, a higher acceptance ratio for FAST-DRONE pushes the network closer to saturation, hence, exhausting the shorter paths as more VNs are embedded. Therefore, FAST-DRONE is forced to choose the longer paths for the later VNs. In case of PAR, the lower acceptance ratio and the lower network utilization still leaves sufficient room in the shorter paths, resulting in shorter embedding paths on average.

4.6 Related Works

In this section, we first discuss the research efforts that address different aspects of survivable virtual network embedding problem. Then we focus our discussion on the works that are closely related to 1 + 1 – ProViNE. Finally, we briefly discuss the known results regarding the hardness of unsplittable flow problem and graph partitioning problem.

Survivable Virtual Network Embedding (SVNE)

Network survivability has been extensively studied in the context of mapping IP links over WDM optical networks [190, 191, 192, 193]. However, unlike VN embedding, IP link mapping over WDM networks assume that the endpoints are already mapped and addresses the issue of provisioning light paths for embedding IP links. In contrast, node embedding is as important as link embedding in VNE, and a coordination between node and link embedding have been shown to increase the acceptance ratio of VNs [55]. Ensuring survivability in the context of VNE was first addressed by Rahman *et al.*, in [69]. They formulated the problem of ensuring survivability in VNs under single substrate link failure as a Mixed Integer Program and proposed heuristics to obtain solutions in a reasonable time. However, unlike 1 + 1 – ProViNE their proposal does not protect VNs from node failures. A number of subsequent research works since then have

addressed different aspects of SVNE such as considering node failures [194, 195, 196, 197, 198], optimizing backup resource allocation [199, 200], and ensuring certain level of availability of the virtual resources [201, 202, 203]. In the rest of this section, we briefly discuss some of the works that consider different aspects of SVNE and contrast them with our proposal.

Some of the early works that consider node failure for SVNE are presented in [194, 195]. Yu *et al.*, addressed the issue of ensuring survivability under regional substrate failures in [194]. A regional substrate failure corresponds to a substrate damage of a facility due to a disaster, leading to multiple substrate node failures. Yu *et al.*, addresses the regional failure scenario by pro-actively provisioning backup nodes at different geographical locations. Subsequent research on ensuring survivability under regional failures, including the ones presented in [196] and [197], propose to take reactive measures after a regional failure and also allow VNs to operate with degraded QoS during regional failures for reducing backup provisioning overhead. The research presented in [195] and [198] address the SVNE problem for a single substrate node failure. Both of these works propose to enhance a VN with additional virtual resources. During a substrate node failure, virtual nodes and links are migrated to the additional resources to restore the VN. In contrast, we take a proactive approach and provide dedicated protection for each component of the VN to facilitate fast recovery.

Some SVNE approaches optimize the backup bandwidth provisioning by sharing the backup path of multiple virtual links [199] or by leveraging multipath embedding of virtual links [200]. However, with a shared backup scheme such as [199], multiple virtual links sharing the same backup can suffer from degraded QoS during a single substrate link failure. Khan *et al.*, proposes to optimize the backup bandwidth provisioning by embedding a virtual link over multiple substrate paths and provisioning a fraction of the virtual link bandwidth instead of the full bandwidth over each such path. The advantage of this approach is that it requires less backup bandwidth and can survive single link failures with full QoS. However, they assume virtual links can be splittable, which is not the case for 1 + 1 − ProViNE.

SVNE problem has also been addressed from the point of view of guaranteeing availability. For instance, the research presented in [201] and [202] have addressed the issue of guaranteeing availability of virtual resources in the context of Virtual Data Centers (VDCs). A VDC is an extension to a VN to include compute, memory and storage resources. Jiang *et al.*, , has addressed has addressed the issue of ensuring VN availability for optical SNs in [203]. These works determine the number of backups required for a virtual resource based on the historical reliability data on the substrate resources. Subsequently, the embedding ensures that the primary and the backups provide a desired level of availability for that virtual resource. In contrast, the number of backup resources are already known for 1 + 1 − ProViNE and we need to find a resource efficient embedding.

Virtual Network Embedding with Dedicated Protection

The motivation for 1 + 1 – ProViNE comes from use cases in T-SDN virtualization, where customers are provided with full-fledged VNs instead of traditional end-to-end connectivity. The work presented in [70] identifies dedicated protection for an entire VN topology as one possible customer requirement for reliability among others. Therefore, it becomes important for the InP to embed a customer VN request with $1 + 1$-protection for the entire topology in a resource efficient way. In this context, the most related to our work is the one by Ye *et al.*, in [3], which addresses the problem of providing dedicated protection for VNE. They formulate the problem using a Quadratic Integer Program in contrast to our ILP formulation. However, the major difference between their approach and ours is the objective. They focused on increasing the VN request acceptance ratio over time, whereas we focus on minimizing the resource allocation cost for embedding VNs. Ye *et al.*, proposed a greedy heuristic based on the node resource requirement, which is not suitable for our case since we do not consider any node capacity or node embedding cost. Lastly, [3] does not consider the location constraint, which is an important constraint in our case. Another closely related work is from Jiang *et al.*, [204], where they propose a backup scheme where the backup virtual nodes are disjoint from the primary virtual nodes. However, the backup nodes can share a single substrate node and therefore exhibit lesser survivability compared to 1 + 1 – ProViNE. They also do not provision full backup of the virtual links, rather provisions some backup paths that can be used to route between the virtual nodes during a single substrate resource failure. More recently, Shahriar *et al.*, has addressed the 1 + 1 – ProViNE problem in the context of Elastic Optical Networks (EONs) [205]. In contrast to our solution, they only consider providing protection to the virtual links and allow for degraded QoS during substrate failure.

Unsplittable Flow and Graph Partitioning

The root of providing dedicated protection for virtual network embedding goes back to combinatorial optimization problems such as graph partitioning and multi commodity unsplittable flow problem [108]. Relevant literature shows that they are computationally hard to solve. Finding a constant factor approximation algorithm for these problems for general graphs is still open [185, 186, 206]. The best known approximation ratio for graph partitioning is not constant, rather it is a poly-logarithm function of the number of nodes [186]. For the unsplittable flow problem with known sources and destinations, the first constant factor approximation algorithm had $(7 + \epsilon)$ and $(8 + \epsilon)$ approximation ratio for simple line graph and cycle graph, respectively [185]. Approximation ratio for the same types of graphs has since been improved to $(2+\varepsilon)$ by Anagnostopoulos *et al.*, [184]. Friggstad *et al.*, has proposed a Linear Programming

relaxation based algorithm for unsplittable flows on trees [207]. However, the approximation ratio for this algorithm is not constant, rather it is a logarithmic function of the number of nodes. Without known sources for the commodity and the flow destinations, this problem is even harder to solve.

4.7 Chapter Summary

In this chapter, we presented solutions for the 1 + 1 − **Pro**tected **Vi**rtual **N**twork **E**mbedding (1 + 1 − ProViNE) problem that embeds a VN on an SN while ensuring dedicated backup for each virtual node and link. We presented DRONE, a suite of solutions for 1 + 1 − ProViNE. We devised an ILP based optimal solution (OPT-DRONE) as well as a heuristic (FAST-DRONE) for solving larger problem instances. Trace driven simulations using both real and synthetic network topologies showed that FAST-DRONE can solve 1 + 1 − ProViNE in a reasonable time frame with only 14.3% extra resources on average compared to OPT-DRONE. Simulation results also showed that FAST-DRONE can accept 3.8× more VN requests compared to state-of-the-art solution [3].

Chapter 5

Adaptive Monitoring of Softwarized Networks

5.1 Introduction

In this chapter, we address a longstanding issue in network monitoring in the context of softwarized networks. Namely, striking a balance between network monitoring overhead and the accuracy of the network view constructed from monitoring data. Constructing an accurate view of a network in a timely manner leaves substantial resource footprint across all the networking planes. For the control plane, constructing an accurate and real-time global network view requires frequently querying the data plane for traffic statistics. A short interval between successive queries increases the accuracy and timeliness of the network view, however, increases control plane bandwidth and message processing overhead at the controller [208]. For the data plane, fine-grained and accurate monitoring of the network packets (*e.g.*, through INT) or flows (*e.g.*, using TCAMs) is typically achieved at the expense of increased data plane bandwidth [102] and memory overhead [89], respectively. In this regard, we aim to identify *less interesting* observations while collecting monitoring data and adapt the monitoring accordingly. Our objective is to reduce both the control and data plane overhead without negatively impacting the quality of collected monitoring data. To this end, we make the following contributions:

- **PayLess**: A traffic intensity-aware variable frequency monitoring algorithm for reducing control plane overhead in OpenFlow network monitoring.
- Evaluation of PayLess on Mininet and comparison with two contemporary approaches, namely, periodic polling and FlowSense [209].

- **LINT**: An accuracy-adaptive and lightweight in-band network telemetry mechanism that runs in the data plane and reduces data plane overhead of network telemetry using live network traffic.
- Evaluation of LINT using a combination of network emulation and simulation and publicly available real network traces.

The rest of the chapter is organized as follows. We first give a brief overview of OpenFlow network monitoring and INT in Section 5.2. Then, we present PayLess, our contributions in reducing the control plane overhead for OpenFlow network monitoring in Section 5.3. We also evaluate PayLess using Mininet network emulation and present the results in Section 5.3. Then, we describe LINT, our work on reducing the data plane overhead of INT in Section 5.4. Here, we motivate the problem by demonstrating the impact of performing network telemetry on live network traffic using some representative data center and WAN packet traces. Then, we present an accuracy-adaptive and lightweight INT mechanism followed by its evaluation in the same section. Finally, we summarize the chapter contributions in Section 5.6.

5.2 Background

5.2.1 OpenFlow Network Monitoring

We first briefly describe how OpenFlow sets up the forwarding path for a new flow in the network. OpenFlow identifies a flow using the fields obtained from layer 2, layer 3 and layer 4 headers of a packet as defined in the OpenFlow specification. When a switch receives a packet that belongs to a flow not matching any rules in the switch's forwarding table, the switch sends a `PacketIn` message to the controller. Upon receiving a `PacketIn` message from a switch, the controller computes the path that the packets belonging to the flow should take leveraging a global view of the network. Then, the controller installs necessary forwarding rules in the switches on the flow's path by sending a `FlowMod` message. The controller can specify an *idle timeout* for a forwarding rule. This idle timeout refers to an inactivity period, after which a forwarding rule (and eventually the associated flow) is evicted from the switch's flow table.

After a flow rule is installed in a switch, there are multiple ways of monitoring the flow. A passive approach is piggybacking the flow table's counter on a `FlowRemoved` message. When a flow is evicted from a switch (*e.g.*, due to idle timeout expiration), the switch sends a `FlowRemoved` message to the controller. This message contains the duration of the flow as well as the number of bytes matching this flow entry in the switch. In addition to this passive approach, the

Table 5.1: Example of telemetry data

Telemetry data	Description
Switch ID	Identifier associated with a switch
Ingress Port ID	Identifier of the port the packet arrived on
Egress Port ID	Identifier of the packet's output port
Ingress Timestamp	The packet's time of arrival
Egress Timestamp	The time when the packet exits the switch
Hop Latency	Time spent by the packet in the device
Egress port TX utilization	Utilization of the packet's output port
Queue occupancy	Queue size when the packet was enqueued for output
Queue congestion status	Fraction of queue being used

controller can also actively query the switch for collecting statistics about a flow table entry. To do so, the controller sends a `FlowStatisticsRequest` message to the switch for querying about a specific flow. In response, the switch sends a `FlowStatisticsReply` message to the controller which contains the duration and the byte count for the requested flows.

5.2.2 In-band Network Telemetry (INT)

INT [94] has recently emerged as a means to obtain per-packet real time view of the network. INT is an outstanding effort to enable network devices (*e.g.*, software and hardware switches, Network Interface Cards (NICs)) to embed device internal state such as packet processing latency, queue depth and link utilization into each passing packet, consequently, facilitating a real-time and microscopic view into network traffic. Such fine-grained telemetry capability is enabling new use-cases such as pin-pointing the root cause of congestion and packet drops through switch queue profiling [95] and per-packet fine-grained feedback to support low-latency data center transport [96], which are otherwise difficult to perform with traditional network monitoring. As of today, INT is supported by commodity hardware such as fixed function and programmable switches [95, 97], and SmartNICs [98, 99], and is being deployed in production telecommunications and data center networks [100, 96].

INT enables network devices to add telemetry information directly to the passing packets with minimal involvement from the control plane. The INT specification makes the following conceptual classification of network devices:

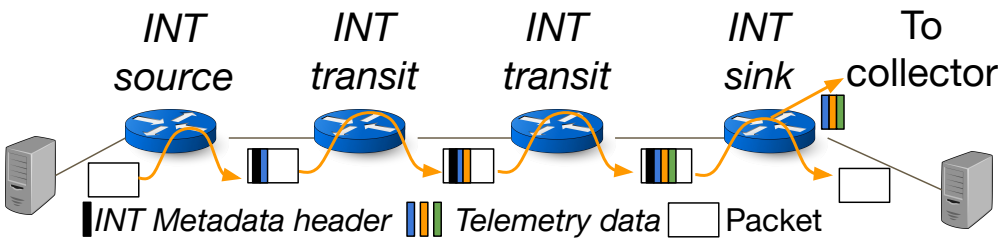


Figure 5.1: INT in action

- **INT source**: This is the first network device on a packet's path that initiates INT. INT source can be a software switch, a SmartNIC, an INT capable top-of-rack switch or a border router.
- **INT transit**: The network devices on a packet's path that embed telemetry information into the packets are called the INT transit nodes.
- **INT sink**: The last INT capable device on a packet's path is the INT sink. An INT sink strips off all the telemetry information from a packet, constructs an INT report according to the INT specification [94] and sends the INT report to a collector. The INT sink can be configured to send all or selectively some of the reports to collector based on predefined policies.

The operation of INT is summarized in Figure 5.1. An INT source can be configured to initiate telemetry data collection for each packet or for packets matching a watchlist. The INT source initiates the telemetry data collection process by encapsulating the packet using one of the protocols described in the INT specification and inserting an *INT metadata header* (12 bytes) to a packet. An INT header contains control information such as the maximum number of INT capable network devices on the packet's path, the encapsulation protocol to use for INT and the set of telemetry data that each INT transit device should add to the packet. Then, each INT transit device adds telemetry data to the packets carrying INT metadata header. Finally, the INT sink strips the telemetry information from the packets, removes the encapsulation and INT metadata headers and restores the original packet before sending it to its destination. The sink can be configured to send all reports to a collector or report only a subset based on pre-defined policies (*e.g.*, report only when total path latency exceeds a threshold).

The operation of INT is summarized in Figure 5.1. An INT source can be configured to initiate telemetry data collection for each packet or for packets matching a watchlist. The INT source initiates telemetry data collection process by by encapsulating the packet using one of

the protocols described in the INT specification and inserting an *INT metadata header* (12 bytes) into a packet. INT header contains control information such as the maximum number of INT capable network devices on the packet's path, the type of encapsulation protocol to use for INT and the set of telemetry data items that each INT transit device should add to the packet. Then, each INT transit device adds telemetry data items to the packets carrying INT metadata header. Finally, the INT sink strips the telemetry information from the packets, removes the encapsulation and INT metadata headers and restores the original packet before sending it to its destination. The sink can be configured to send all reports to a collector or report only a subset based on pre-defined policies (*e.g.*, report only when total path latency exceeds a threshold).

The current INT specification defines a set of telemetry data (4 bytes each) as presented in Table 5.1 [94]. However, programmable switches and SmartNICs enabled by the Protocol Independent Switch Architecture (PISA) [210] can be programmed using the P4 programming language [24] for computing other functions on the packets and the flows (*e.g.*, mean packet size, moving average of queue occupancy), and augment the packets with the result. Telemetry data collected through INT can be used for answering network monitoring queries that require per-packet information (*e.g.*, identifying flows that have a congested switch on its path, computing per-packet end-to-end latency distribution, per-switch queue profiling for identifying root cause of congestion or increased tail latency, among others) or provide feedback to control and management applications (*e.g.*, congestion control [96]).

5.3 PayLess: Adaptive Monitoring from the Control Plane

In this section, we present PayLess, an adaptive monitoring algorithm that adjusts the frequency of collecting network statistics from the SDN control plane based on the intensity of network traffic. Our goal is to collect accurate and timely statistics from the network without incurring a substantial overhead. We assume that the controller uses OpenFlow protocol for querying the switches and collecting statistics from them as described in Section 5.2.1. Although we present our solution in the context of OpenFlow, however, this is applicable to any scenario where a logically centralized and remote control plane collects statistics from the data plane. In the following, we first describe the PayLess algorithm in Section 5.3.1. Then, we present a concrete implementation of the algorithm for a link utilization monitoring use case in Section 5.3.2. Finally, we present our evaluation of PayLess in Section 5.3.3.

5.3.1 The Monitoring Algorithm

A straw man approach for flow statistics collection is to periodically poll the switches by sending the `FlowStatisticsRequest` message. A high frequency polling (*i.e.*, short interval between consecutive queries) will generate a near real-time view of the active flows. However, this will generate a substantial number of control messages, consuming network bandwidth between the data and control plane, and increasing the message processing load on the controller. To strike a balance between statistics collection accuracy and incurred network overhead, we propose a variable frequency flow statistics collection algorithm.

We propose that the SDN controller maintains a table $\mathcal{A}$ in the control plane for tracking the active flows. When the first packet of a flow arrives at a switch and the switch sends a `PacketIn` message, the controller adds a new flow entry to $\mathcal{A}$ along with an initial statistics collection timeout τ. If the flow expires from the switch within the next τ time units, the controller will receive the expired flow's statistics in a `FlowRemoved` message. Otherwise, a timeout event will be triggered in the controller after τ time units and the controller sends a `FlowStatisticsRequest` message to the corresponding switch for collecting that flow's statistics (*e.g.*, packet and/or byte count). After collecting statistics of a flow from the switch, the controller adjusts the timeout τ of that flow based as follows:

- If that flow's statistics (after collecting from the switch) did not significantly change within the last τ time period (*i.e.*, the difference between the previous and current value is below a threshold Δ_1), its statistics collection timeout is multiplied by a factor α. For a flow with low packet or byte rate, this process will be repeated until a maximum timeout value of $\mathcal{T}_{max}$ is reached.

- If the collected statistics changed significantly since the last query, *i.e.*, the difference is more than Δ_2, the statistics collection timeout of that flow is divided by a factor β. For a flow, that is significantly changing, this process may be repeated until a minimum timeout value of $\mathcal{T}_{min}$ is reached.

The rationale behind this timeout adjustment is that flows that significantly changed in volume (bytes or packets) have higher chances of contributing to triggering interesting network events (*e.g.*, increase link utilization or become heavy-hitters). Therefore, once we detect such change, we increase the corresponding flow's polling frequency. For a similar reason, we reduce the polling frequency of the steady flows since they are not conveying much information at the time. If their contribution changes then the algorithm will adapt the polling frequency accordingly.

We further optimize the proposed algorithm by batching multiple `FlowStatisticsRequest` messages together for flows with the same timeout. In this way, we manage to reduce the spread of monitoring traffic in the network without affecting the effectiveness of polling with a variable frequency. We present the pseudo-code of the described algorithm in Algorithm 6.

Algorithm 6: PayLess algorithm

Input: $\mathcal{A}$: Active flows table at the controller; $\mathcal{S}$: association between statistics collection timeout and active flows (indexed by timeout value)

function ScheduleFlowStatisticsCollection*Event e*
  if e *is* Initialization *event* **then** $\mathcal{A} \leftarrow \phi, \mathcal{S} \leftarrow \phi, U \leftarrow \phi$
  if e *is a* `PacketIn` *event* **then**
    $f \leftarrow \langle e.switch, e.switch\_port, \mathcal{T}_{min}, 0 \rangle$
    $\mathcal{S}[\mathcal{T}_{min}] \leftarrow \mathcal{S}[\mathcal{T}_{min}] \cup \{f\}$
  else if e *is a STATISTICS_COLLECTION_TIMEOUT event* **then**
    $\tau \leftarrow e.timeout$
    foreach *flow* $f \in \mathcal{S}[\tau]$ **do** send `FlowStatisticsRequest` to $f.switch$
  else if e *is a* `FlowStatisticsReply` *event for flow* f **then**
    $\delta_{bytes} \leftarrow e.byte\_count - f.byte\_count$
    $\delta_{duration} \leftarrow e.duration - f.duration$
    $checkpoint \leftarrow GetCurrentTimestamp()$
    $U[f.port][f.switch][checkpoint] \leftarrow \langle \delta_{bytes}, \delta_{duration} \rangle$
    if $\delta_{bytes} < \Delta_1$ **then**
      $f.\tau \leftarrow \min(\alpha f.\tau, \mathcal{T}_{max})$
      Move f to $\mathcal{S}[f.\tau]$
    else if $\delta_{bytes} > \Delta_2$ **then**
      $f.\tau \leftarrow \max(\frac{f.\tau}{\beta}, \mathcal{T}_{min})$
      Move f to $\mathcal{S}[f.\tau]$

5.3.2 Implementation: Link Utilization Monitoring

As a concrete use case of our proposed monitoring algorithm, we implement a prototype link utilization monitoring application on Floodlight SDN controller [211]. The source code of the modified Floodlight controller is made available at [212]. Furthermore, without loss of generality, we assume the flows are identified by their source and destination IP addresses.

We intercept the `PacketIn` and `FlowRemoved` messages for keeping track of flow installations and removals from the switches, respectively. We also maintain a hash table indexed by

the schedule timeout value, where each bucket with timeout τ contains a list of active flows that should be polled every τ milliseconds. Each hash table bucket is assigned a worker thread that wakes up every τ milliseconds and sends a `FlowStatisticsRequest` message to the switches corresponding to the flows in the bucket. The `FlowStatisticsReply` messages are received asynchronously by the monitoring module. The latter creates a measurement checkpoint for each received `FlowStatisticsReply` message. The contribution of a flow is calculated by dividing its differential byte count by the time duration from the previous checkpoint. The monitoring module examines the measurement checkpoints of the link being monitored and updates the utilization at the previous checkpoints if necessary. The active flow entries are moved around the hash table buckets with lower or higher timeout values depending on timeout adjustment described in Section 5.3.1. Currently, we provide a REST API for obtaining link utilization information of all links in the network.

5.3.3 Evaluation

We evaluate the link utilization monitoring application built using PayLess algorithm through network emulation on Mininet [109]. We have also implement two other approaches for link utilization monitoring, namely, FlowSense [209] and periodic polling. FlowSense proposes a zero-cost link monitoring algorithm that relies on the flow statistics piggybacked on the `FlowRemoved` messages for constructing a view of link utilization at the SDN control plane. In contrast, in the periodic polling approach, the controller polls the switches at a constant interval to gather link utilization information. In the following, we first describe the experimental setup followed by the evaluation metrics. Finally, we discuss the evaluation results.

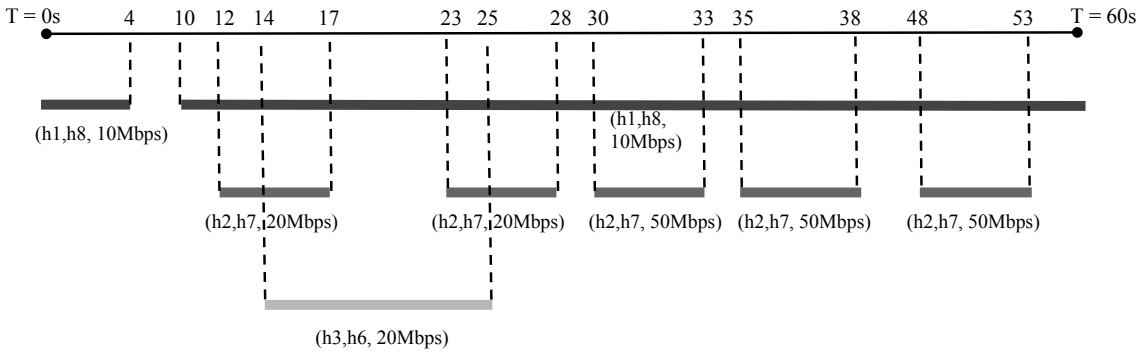


Figure 5.2: Traffic mix for PayLess evaluation

Experiment Setup

We deployed a 3-level tree topology on Mininet as shown in Figure 5.3. We used *iperf* for generating UDP flows for a total duration of 100s between the end hosts. The end-points, arrival and departure time, and goodput of the UDP flows used are shown in Figure 5.2. We have set the idle timeout of the flows in a switch to 5s. We have also deliberately introduced pauses of different duration between the flows in the traffic to experiment with different scenarios. Pauses less than the idle timeout were placed between the 28th and the 30th second, and also between the 33 and the 35 seconds to observe how PayLess and Flowsense react to sudden traffic spikes. The minimum and maximum polling interval for PayLess were set to 500 ms and 5 s, respectively. For the periodic polling case, the polling interval was set to 1 s. The parameters Δ_1 and Δ_2 described in Section 5.3.1 were set to 100 MB. Finally, we have set α and β described in Section 5.3.1 to 2 and 6, respectively. β was set to a higher value for quickly reacting to and adapting to the changes in traffic.

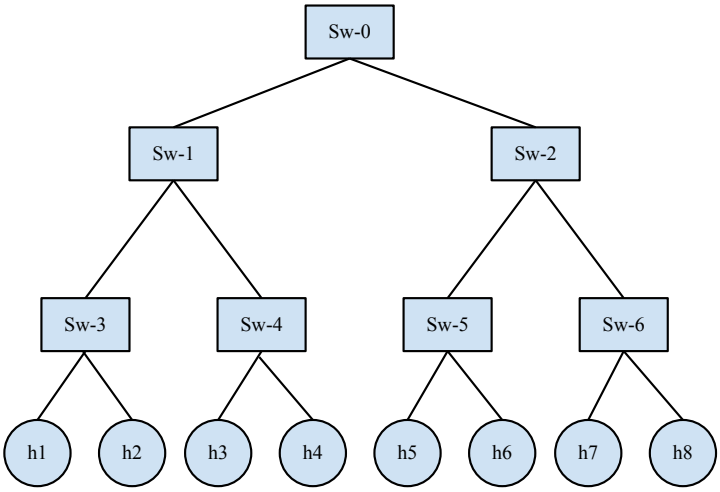


Figure 5.3: Network topology for PayLess evaluation

Evaluation Metrics

Utilization Link utilization is measured as the instantaneous throughput obtained from that link, normalized by the link's capacity. We report the utilization of the link between switches `Sw-0` and `Sw-1` from Figure 5.3. According to the traffic mix, this link is part of all the flows and is the most heavily used. We also experiment with different values of minimum polling interval ($\mathcal{T}_{min}$) and show its effect on the trade-off between accuracy and monitoring overhead.

Overhead We compute overhead in terms of the number of `FlowStatisticsRequest` messages sent from the controller. We compute the overhead at timeout expiration events when flows with the same monitoring timeout are queried together for statistics. When applicable, we report PayLess's overhead normalized by the overhead of the periodic polling approach.

Results

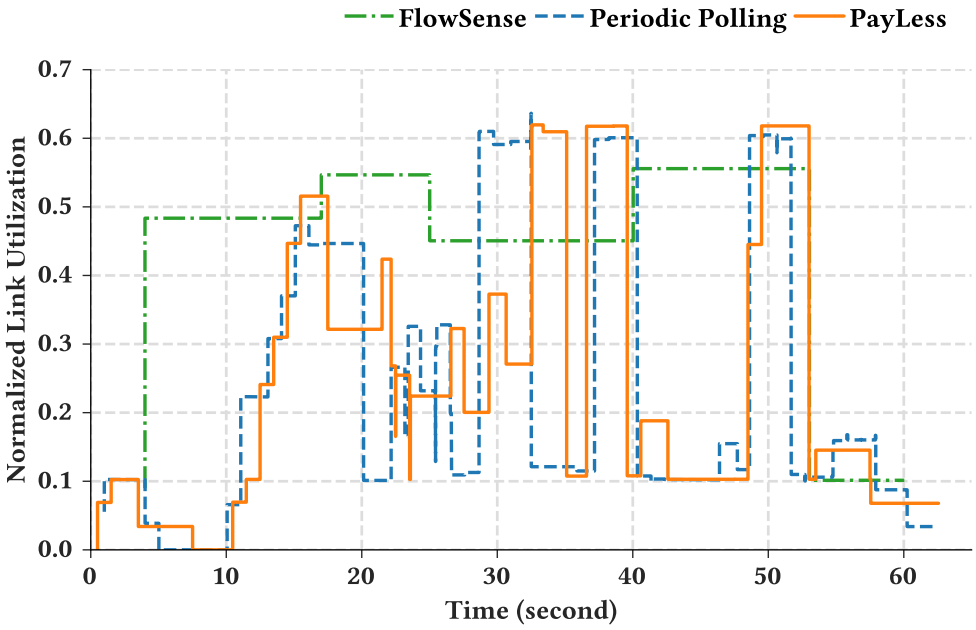


Figure 5.4: Link utilization measurement

Utilization Figure 5.4 shows the utilization of the `Sw0-Sw1` link over the emulation duration, measured using three different techniques. The baseline scenario, *i.e.*, periodic polling has the most resemblance with the traffic pattern used (Figure 5.2). Flowsense fails to capture the traffic spikes because of its coarse grained measurement approach. The traffic pauses less than the idle timeout value cause Flowsense to report less than the actual utilization. In contrast, PayLess very closely follows the utilization pattern obtained from periodic polling. Although PayLess did not succeed to fully capture the first spike in the traffic, the algorithm quickly adjusted to successfully capture the subsequent traffic spikes. We also compute the normalized root mean squared error (NRMSE) (RMSE normalized by the range of utilization values) on a rolling basis over the course of emulation and found the quartiles to be 4.2%, 13%, and 18.4%, respectively.

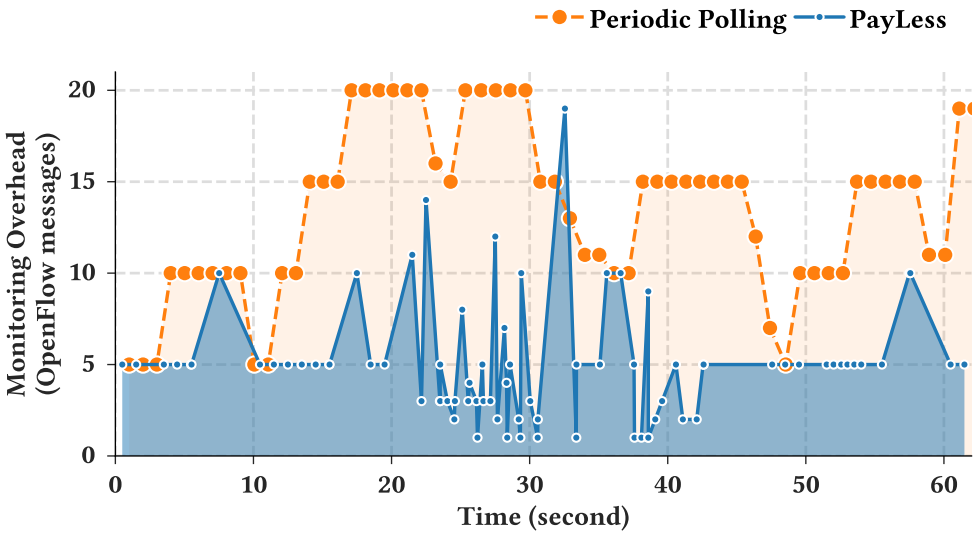


Figure 5.5: Control plane messaging overhead

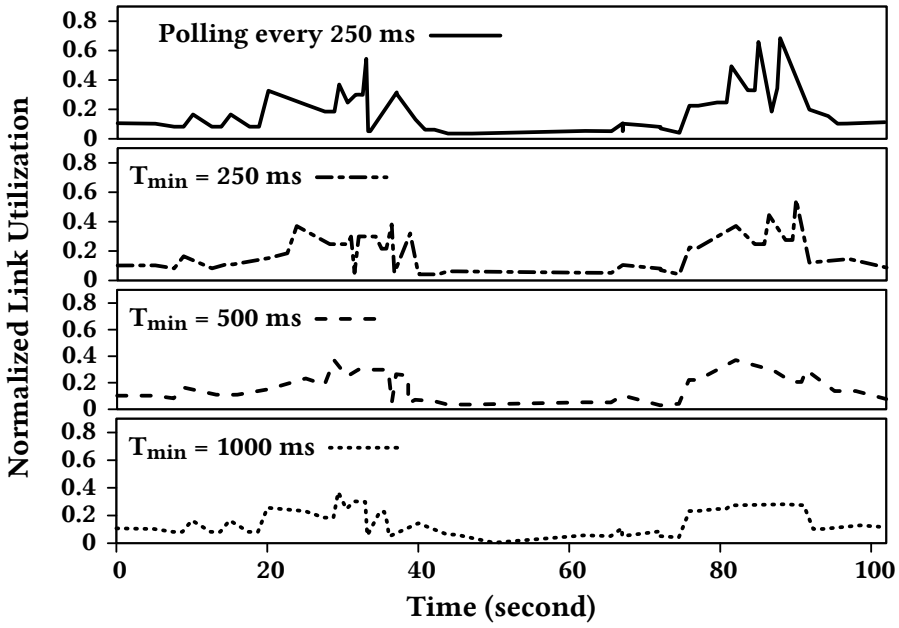


Figure 5.6: Effect of $\mathcal{T}_{min}$ on measured link utilization

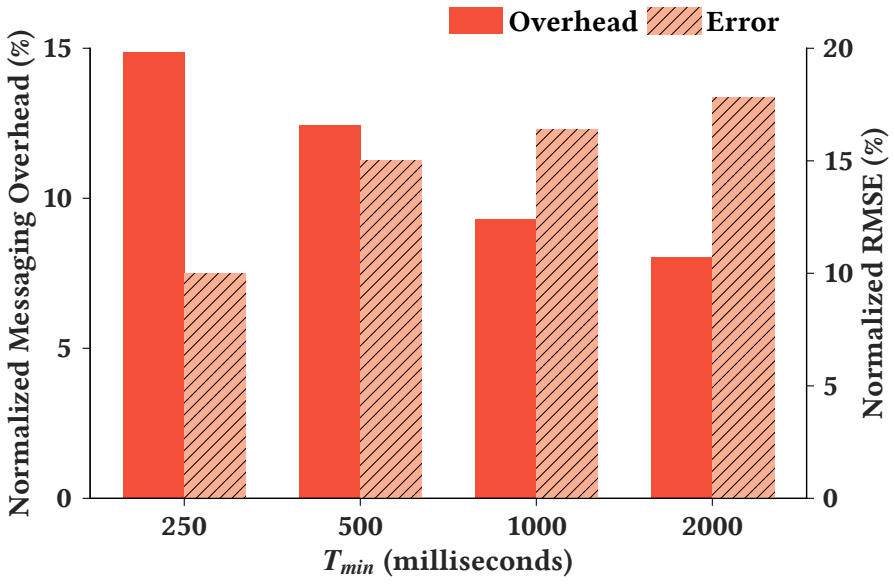


Figure 5.7: Overhead and measurement error

Overhead Figure 5.5 shows the messaging overhead of the baseline periodic polling approach and PayLess. Since Flowsense does not actively send `FlowStatisticsRequest` messages for statistics collection, it has zero messaging overhead. The periodic polling approach polls all the active flows after the fixed timeout expires. This causes a large number of messages to be injected in the network at each query time. In contrast, PayLess reduces the number of `FlowStatisticsRequest` messages sent (consequently the number of `FlowStatisticsReply` messages) by assigning different timeouts to flows and spreading the messages over time. It is also evident from Figure 5.5 that PayLess has more query points across the timeline. However, during each query time, PayLess sends out lesser number of messages to get flow statistics compared to that of the periodic polling approach. Over the course of emulation, we found PayLess to be sending 50% less messages compared to the periodic polling approach.

Although Flowsense has zero measurement overhead, it is significantly less accurate compared to PayLess. In addition, the monitoring traffic incurred by PayLess is very low, only 6.6 messages per second on average, compared to 13.5 messages per second on average for periodic polling in our experiment scenario. In summary, the adaptive approach taken by PayLess can achieve an accuracy close to that of the periodic polling approach, while substantially reduce the messaging overhead.

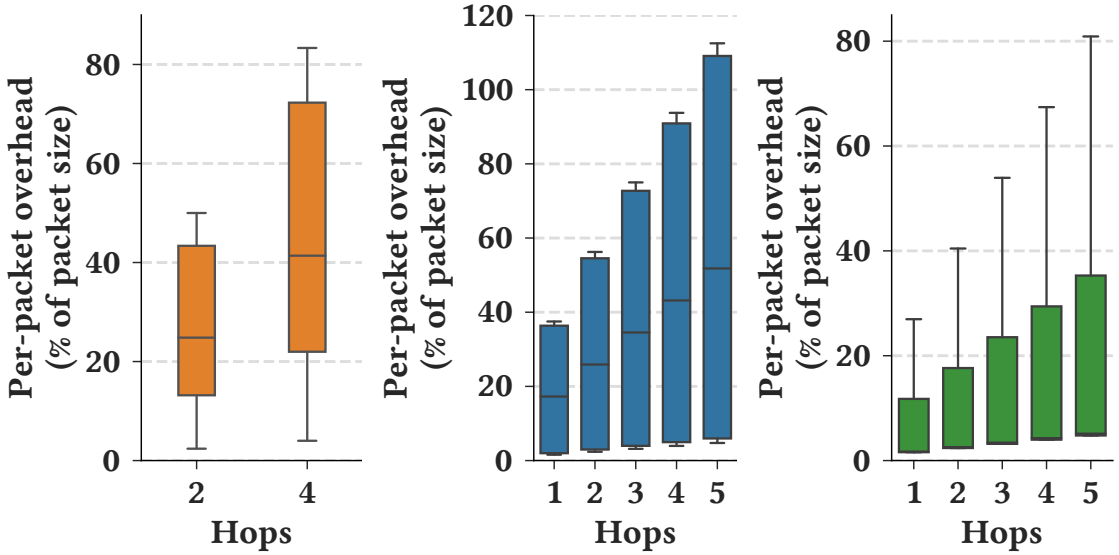
Effect of Minimum Polling Frequency $\mathcal{T}_{min}$ As explained in Algorithm 6, PayLess adapts the monitoring frequency to changes in traffic intensity. For a rapidly changing flow, the poling interval sharply decreases (*i.e.*, polling frequency sharply increases) and reaches $\mathcal{T}_{min}$. In Figure 5.6, we present the impact of $\mathcal{T}_{min}$ on monitoring accuracy (we scale the time axis by duplicating the traffic pattern in Figure 5.2). As the baseline, we consider the time series of monitoring data obtained by periodically polling the flow statistics every 250 ms. Evidently, monitoring accuracy remains the highest, *i.e.*, the least distortion is observed on the time series graph, for $\mathcal{T}_{min}$ = 250 ms. With higher values of $\mathcal{T}_{min}$, monitoring accuracy gradually degrades as can be observed by the distorted shape of the time series. However, monitoring accuracy comes at the cost of network overhead, which we present in Figure 5.7. This figure presents the NRMSE of the obtained monitoring data compared to periodic polling, computed over all the measurements obtained during emulation. We also show the incurred messaging overhead for different values of $\mathcal{T}_{min}$ in Figure 5.7, normalized by that of the periodic polling approach. The $\mathcal{T}_{min}$ parameter can be adjusted to trade-off accuracy with messaging overhead, depending on the application requirements.

5.4 LINT: Accuracy-adaptive INT from the Data Plane

In this section, we aim at reducing the data plane overhead of INT while reaping its benefits as much as possible. Our objective is to identify and filter the *less interesting* observations of telemetry data directly in the data plane without negatively impacting the quality of collected telemetry data. We use the quality of results produced by different network monitoring queries as an indicator of the quality of the telemetry data. To this end, we present LINT, an accuracy-adaptive and **L**ightweight **INT** mechanism that runs in the data plane. Network devices employing LINT independently decide on selectively reporting telemetry data on a passing packets, without any explicit coordination and intervention from a control plane. In the following, we first present an empirical study demonstrating the extent of incurred INT data plane overhead using real network traces. Then, we present our solution in Section 5.4.2 followed by the evaluation of LINT using real network traces in Section 5.4.3.

5.4.1 Motivation

There are several sources of overhead in INT. First, the INT source adds a 12 byte INT metadata header to each packet [94]. Then, each INT transit node adds one or more telemetry data items to the packet according to the instruction embedded in the INT metadata header. The INT specification reserves 4 bytes for each telemetry data item to be added [94]. Therefore, a switch

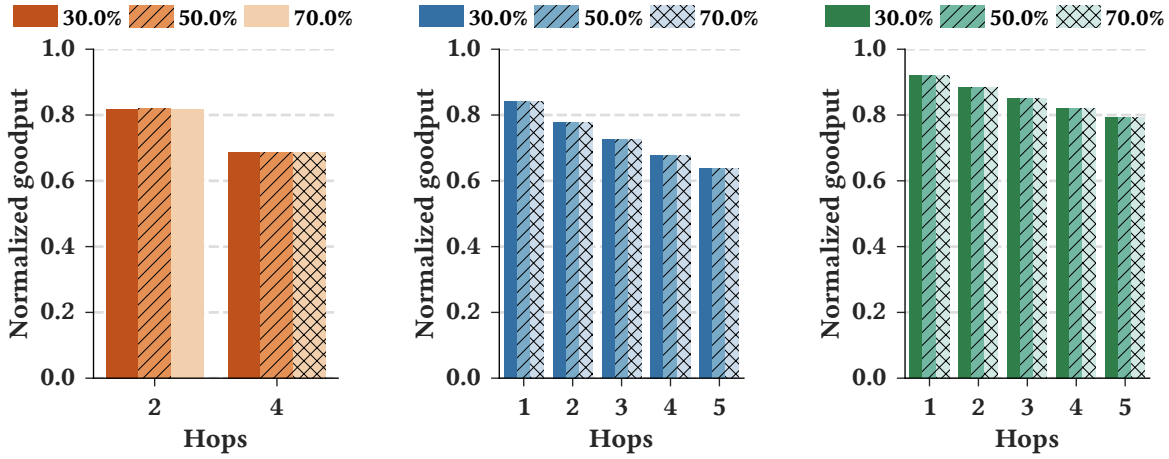


(a) FB-WS [213] (b) UNIV1 [214] (c) CAIDA (Equinix 2016) [215]

Figure 5.8: Packet size increase due to INT

reporting 3 data items such as its SwitchID, the hop latency and the queue occupancy will add 12 bytes to the passing packets. Since each transit node on a packet's path adds telemetry data to the packet, INT overhead increases linearly with the path length of the packet. For instance, if a packet goes through 5 INT switches (including the INT source) and each switch is adding 3 telemetry data items, then the packet will have 72 bytes added to it when it reaches the INT sink. For MTU size packets (1500 bytes for Ethernet), the bit overhead can translate to ≈5% increase in packet size. However, the mean and median packet sizes in most networks are typically much smaller than the MTU (*e.g.*, median packet size is close to 250 bytes in data centers [214, 213]), resulting in higher per-packet bit overhead due to INT.

More recently, an empirical study in [102] sheds light on the extent of network goodput degradation due to INT's bit overhead in the data plane. This study used ns3 simulation on a fat-tree data center network [216] with synthetic traffic generated from a web search workload. The results show that INT's bit overhead in the data plane can reduce network goodput by as much as $\approx 20\%$. In this section, we present a complementary study to that presented in [102], demonstrating the extent of per-packet bit overhead on some publicly available real network traces. Specifically, we analyze the traces from Facebook's production web service cluster (FB-WS) [213], a campus data center network (UNIV1 data set from [214]) and a wide-area network traffic capture from CAIDA (Equinix 2016) [215]. Unlike the UNIV1 and CAIDA traces, the



(a) FB-WS (b) UNIV1 (c) CAIDA

Figure 5.9: Mean normalized goodput of a link by varying the INT hops (*i.e.*, the per-packet overhead) and link utilization (considering median packet sizes from different network traces)

FB-WS trace does not contain packet captures. Rather, it contains meta-data extracted from sampled packets (*e.g.*, packet size, source and destination racks and pods) from the end-hosts.

The result of our analysis is presented in Figure 5.8. For this analysis, we assumed collecting three telemetry data items from each switch on a packet's path. We varied the number of INT devices on a packet's path (*i.e.*, Hops in the figure) from 1 to 5. To provide some context, in a fat-tree data center network [216], a path between servers within the same pod and different pods is 2 and 4 hops, respectively. For wide area networks such as the autonomous system level graph, the median path length lies between 5 and 6 [217]. Note that for the FB-WS trace, we leveraged the meta-data describing a packet's source and destination pod and rack, and correlated that with Facebook's data center network architecture [218] to identify the number of hops. All the clusters in Facebook's production network exhibited similar behavior, except for the Hadoop cluster, where almost all the packets were MTU sized. For the other traces, no such meta-data was made available, hence, we experimented with a range of hop counts. In Figure 5.8, we plot the distribution of per-packet overhead (the box represents the quartiles and the bars' end points represent the extremes) in terms of percent increases from original packet size.

For the FB-WS, UNIV1, and CAIDA traces, the median packet size increase is ≈40%, ≈40%, and ≈10% compared to the original packet size, respectively, for collecting three telemetry data

items on a 4-hop path. On the higher end of the distribution, we observe the 75th percentile overhead under the same setting to be ≈70%, ≈90%, and ≈30%, respectively, which is significant. A direct consequence of packet size increase for transporting telemetry data is reduced network goodput since lesser fraction of the bits in a packet remain available for transporting the original network traffic.

We also conduct another analytical study to measure the impact of INT data plane overhead on network goodput. For this study, we consider a 10 Gbps network link carrying traffic similar to that of the UNIV1 trace [213]. We assume the packet sizes to be uniform and equal to the median packet size in UNIV1 trace. With this assumption, we compute that link's goodput considering different levels of utilization and normalize the result with that from the case when INT is not used. We can see from the results in Figure 5.9 that even for one hop, INT data plane overhead can reduce network goodput by ≈20%. Moreover, on a typical pod-to-pod path in a fat-tree topology (*i.e.*, 4 hops), the goodput can reduce by ≈30% due to packet size increase. These analytical results motivate the need for mechanisms that can strike a balance between INT data plane overhead and the quality of collected telemetry data for answering network monitoring queries.

5.4.2 The LINT Algorithm

We can draw an analogy between INT and sensor networks. INT capable devices are similar to sensors that measure and report certain metrics from the environment to a collector or sink. Sensors are typically resource constrained (*e.g.*, limited battery life and limited network bandwidth), therefore, need to carefully measure and report without abusing the constrained resources. Albeit not constraint, however, INT needs to work in a way to not incur significant data plane overhead, thereby, negatively impacting regular network operations. Given the similarity, we leverage techniques from *model-driven data acquisition*, a well-studied topic in the sensor networking literature for reducing data transmission from sensor nodes to the sinks [219]. We propose LINT, an accuracy-adaptive and lightweight INT mechanism that can run in programmable data plane. LINT selectively reports telemetry data items on passing packets by estimating accuracy loss at the collector. LINT can be implemented within the constraints of commodity programmable PISA devices, and can work without any global coordination and intervention from a control plane.

In the following, we first give a brief overview of model-driven data acquisition that forms the basis of our solution followed by the description of LINT. We also present LINT-flow, an extension of LINT that takes the flow-context of the packets into consideration. Finally, we conclude this section with some implementation considerations for LINT.

Overview of Model-driven Data Acquisition

In the model-driven data acquisition paradigm, a prediction model is used to determine if sensors should be queried for new data or to filter the data at the sensor level. The prediction model is devised for capturing the pattern of the measurements or for correlating measurements of different metrics. At one extreme of the approach is the one presented in [219], where the model is solely used for determining if a query engine should query the sensor nodes for measurements or not. The query engine queries a sensor only when it determines that the model output is not sufficient for maintaining a satisfactory level of accuracy. In contrast, the Spanish Inquisition Protocol (SIP) [220] is on the other extreme where a predictor is used to solely determine if a sensor reading should be transmitted to a sink or can be dropped at the sensor level. The sensors using SIP use a predictor function to forecast what the sink is expecting next to receive. When the estimation indicates that the current sensor reading is far off from what the sink is expecting, only then the sensors send their readings to the sink. Our problem is close to the latter, *i.e.*, determining from the data plane if telemetry data items should be reported. Hence, we will use SIP as a basis for our solution. A comprehensive survey of model-driven data acquisition techniques can be found in [221].

LINT: Accuracy-adaptive and Lightweight INT

Overview While designing LINT, our goal is to keep it lightweight. In other words, LINT should be capable of running within the constraint of commodity PISA devices such as no floating point operations, limited to no multiplication and no division, no loops or recursion, limited number of match-action stages and limited match-action entries per-stage, and only one stateful register memory read-modify-write per packet processing stage [210, 222]. Furthermore, executing LINT should not consume substantial amount of device resources and should leave enough resources for running other applications (*e.g.*, [223, 224, 225, 226]) on a PISA device.

Therefore, to keep LINT simple, we build on SIP presented in [220]. For each packet with an INT metadata header that arrives at a PISA device, LINT makes the decision of reporting a telemetry data item according to Algorithm 7 as follows. A device running LINT tries to estimate the amount of error that can be introduced at the collector if the requested telemetry data items are not piggybacked on the current packet. For estimating this error, the device uses a predictor function for each telemetry data item of interest. The predictor function for a telemetry data item d is used for computing the following:

- d_D^{next}: the predictor function applied on all past observations of d in this device.
- d_C^{next}: the predictor function applied on the observations of d reported to the collector.

Algorithm 7: LINT algorithm

Input: p = The current packet; $\mathcal{D}$ = metrics to monitor; α = weight parameter of EWMA; δ = error threshold

function LINT$(p, \mathcal{D}, \alpha, \delta)$
  foreach $d \in \mathcal{D}$ **do**
    $val_d \leftarrow$ current observation of d
    $s \leftarrow d_D^{next}$, $t \leftarrow d_C^{next}$
    $d_D^{next} \leftarrow \alpha val_d + (1-\alpha)s$
    $deviation \leftarrow |d_D^{next} - t|$
    if *deviation* $> \delta d_D^{next}$ **then**
      $p.add\_telemetry\_observation(d, val_d)$
      $d_C^{next} \leftarrow \alpha val_d + (1-\alpha)t$

Essentially, the quantity d_C^{next} denotes what the collector will predict about the observation of d if the current observation is not reported. The device decides to report the currently observed value of d if the difference between d_D^{next} and d_C^{next} is within an operator defined fraction δ of d_D^{next}, *i.e.*, $|d_D^{next} - d_C^{next}| \leq \delta \times d_D^{next}$. In other words, when the device estimates that the prediction error at the collector can go above an acceptable threshold, it reports the current observation. Otherwise, the device skips reporting the current observation of d. In this way, LINT adapts telemetry data reporting to estimated error. Note that we can choose the parameter δ to be in the form 2^{-m}, in this way replace the multiplication operation by a bit shift operation.

Device and collector coordination The value of d_D^{next} is updated whenever a packet arrives with INT metadata header instruction for reporting d. However, d_C^{next} is updated only when the device reports the an observation of d piggybacking on a packet. The collector replaces any missing telemetry data item d not reported by a device on the packet's path by using the same predictor function as the device. In this way, both the device and the collector stay in sync about the extent of the error due to not reporting an observation of a telemetry data item d. Also, each device independently makes their own decision. Indeed, additional information about the error estimate can improve the quality of decision making for a device. However, that would require coordination between the devices and is not a desirable for keeping LINT lightweight.

Choice of predictor function

The concrete realization of LINT requires deciding on a predictor function that can be computed within the constraints of PISA devices. In this regard, we chose from the moving average family of predictor functions since they have a constant memory footprint, have less number of parameters to tune and are computationally lightweight. We leave the exploration of more computationally demanding predictor such as machine learning based prediction [227] for a future exploration. Specifically, we chose to use Exponentially Weighted Moving Average (EWMA) [228] for LINT. EWMA computes moving average of a data stream by applying exponentially decaying weights to the items in the stream according to the order they appear. As time progresses, observations further in the past have lesser and lesser impact on EWMA. We can recursively compute EWMA for a stream of observations $\tilde{x} = \langle x_0, x_1, \ldots x_t \rangle$ as follows:

$$S_0 = x_0$$
$$S_t = \alpha x_t + (1 - \alpha) S_{t-1} \quad (0 < \alpha < 1)$$

Here, x_t is the observation at the current time t, S_t is the EWMA at time t computed from x_t and S_{t-1}. The weight α determines how much importance will be given to the past observations. The multiplication term involving the fraction α can be avoided by choosing α of the form 2^{-m} (for some integer $m > 0$) [222]. By doing so, we can rewrite the EWMA computation equation as follows:

$$S_t = S_{t-1} + 2^{-m}(x_t - S_{t-1}) \quad (m > 0) \tag{5.1}$$

The multiplication by 2^{-m} ($m > 0$) in (5.1) can be performed by shifting bits to the right m times, which is supported by commodity programmable hardware.

LINT-Flow: Flow-context aware LINT

Very often packets from the same network flow exhibit similar behavior (*e.g.*, often due to packets of the same flow belonging to the same application) and are subjected to same operational policies (*e.g.*, packets from the same flow sent to the same output queue based on flow priority). Therefore, applying the predictor function with a packet's flow context in consideration has the potential to reduce errors. In this regard, we propose LINT-flow, an extension of LINT that also takes a packet's flow context into consideration while applying the predictor.

In contrast to maintaining a pair of EWMA values for each telemetry data item (*i.e.*, EWMA of all observations in a device and EWMA of the observations reported to the collector from the device), we maintain a pair of EWMA values for each observed flow in a hash table. Without loss

Algorithm 8: LINT-flow algorithm

Input: p = The current packet; $\mathcal{D}$ = telemetry data items of interest; $\mathcal{F}^D_{\mathcal{D}}$ = table for per-flow EWMA values of all the observations for telemetry data items in $\mathcal{D}$; $\mathcal{F}^C_{\mathcal{D}}$ = table for per-flow EWMA values of the previously reported observations for telemetry data items in $\mathcal{D}$; α = weight parameter of EWMA; δ = error threshold

function LINT$(p, \mathcal{D}, \mathcal{F}^D_{\mathcal{D}}, \mathcal{F}^C_{\mathcal{D}}, \alpha, \delta)$
  $flow\_key \leftarrow \langle p.src\_ip, p.dst\_ip, p.nw\_proto, p.src\_port, p.dst\_port \rangle$
  foreach $d \in \mathcal{D}$ **do**
    $val_d \leftarrow$ current observation of d
    $s \leftarrow \mathcal{F}^D_d[flow\_key], t \leftarrow \mathcal{F}^C_d[flow\_key]$
    $d^{next}_D \leftarrow \alpha val_d + (1-\alpha)s$
    $\mathcal{F}^D_d[flow\_key] \leftarrow d^{next}_D$
    $deviation \leftarrow |d^{next}_D - t|$
    if *deviation* $> \delta d^{next}_D$ **then**
      $p.add\_telemetry\_observation(d, val_d)$
      $\mathcal{F}^C_d[flow\_key] \leftarrow \alpha d_{current} + (1-\alpha)t$

of generality we assume the network flows are identified by the five tuple (source IP, destination IP, network protocol, source port, destination port). When a packet with INT metadata header arrives at a device, LINT-flow identifies the hash table entries corresponding to the flow that the packet belongs to and updates the EWMA values accordingly. Subsequently, LINT-flow considers the difference between the EWMA values corresponding to the packet's flow while deciding which telemetry data item(s) should be reported on a packet. The LINT-flow algorithm is presented as pseudocode in Algorithm 8.

Implementation Considerations

Realizing LINT on programmable data plane will require changes to the INT protocol message formats. One key issue that must be addressed is how to communicate to the collector that only a subset of the originally requested telemetry data items have been reported. One solution is to embed a bitmap at each INT transit node, representing the telemetry data items that the node is reporting. To avoid consuming more bits, this bitmap can share unused space in other fields such as the SwitchID. Devising a robust solution for this issue requires further investigation and we leave it for future exploration.

Our ongoing implementation effort is mostly simulation-centric and in part is around bmv2,

the P4 reference software switch. A full-fledged implementation on a programmable PISA hardware is yet to be done. In this section, we briefly describe the potential data plane resource requirements for LINT. For the LINT algorithm, we need two stateful registers per telemetry data item for maintaining the EWMA of the observations at the device and the EWMA of the observations sent to the collector. Therefore, a total of $2|\mathcal{D}|$ processing stages and $2|\mathcal{D}|$ register entries will be needed for dealing with a set telemetry data items $\mathcal{D}$. For instance, to selectively report hop latency and queue occupancy (*i.e.*, $\mathcal{D} = \{\text{hop latency, queue occupancy}\}$), we will require 4 processing stages and 8 register entries in total. Finally, the parameters α and δ are not expected to change very frequently, therefore, we can specify them as constants during pipeline configuration.

The limited number of memory access per processing stage (typically 1 for known hardware targets [226]) pose a problem in conditionally updating the EWMA of the observations sent to the collector. Once this EWMA value is accessed from the corresponding register entry, the register entry cannot be accessed again in the same or subsequent processing stage. This limitation can be addressed by recirculating a packet with EWMA update (if needed) within the switch and update the corresponding register entry while processing the recirculated packet.

For the LINT-flow algorithm, we will need $4|\mathcal{D}|$ processing stages considering a flow cache implementation similar to that in [226]. However, at each processing stage we will require a hash table that can be implemented using a register array for keeping track of the active set of flows and their corresponding EWMA values. Indeed, keeping track of all flows per processing stage will be impractical. However, one observation is that most network flows are short-lived, especially in data centers [214, 213]. Therefore, the active set of flows will be changing fast, which creates the opportunity for applying cache eviction policies to track only a subset of flows at a time. We present a simulation to study demonstrating the impact of tracking a limited number of flows per processing stage in Section 5.4.3.

5.4.3 Evaluation

We employ a combination of network emulation and simulation to evaluate the effectiveness of LINT and LINT-flow. Before describing the evaluation results we first briefly describe the methodology. The goal of our evaluation is to contrast between different aspects of LINT with that of performing INT for each packet in the network. To accomplish this, we first deploy a network consisting of bmv2 switches (P4 software switch) using Mininet. The bmv2 switches run a P4 program that implements INT (a modified version of the *int.p4* implementation provided with the ONOS SDN controller). After subjecting the deployed network with traffic through different hosts, we collect the generated INT reports by passively capturing packets on relevant

INT sink switch interfaces. These INT reports provide us with the ground truth to compare against. Then, we simulate LINT and LINT-flow on the on the captured INT reports to obtain modified INT reports that LINT and LINT-flow would generate when deployed in the network. These generated INT reports are then used for executing several network monitoring queries and the results are compared against the query results obtained using the ground truth. Since network emulation does not provide predictable and reproducible timing behavior, we cannot reproduce the same per-packet latency and queue occupancy in the switches in successive runs and compare between approaches, hence, our hybrid approach.

In the following, we first describe the setup and the evaluation metrics. Then we present our evaluation results focusing on the following scenarios: (i) evaluation of INT data plane overhead reduction by using LINT; (ii) evaluation of the impact of selectively reporting telemetry data items by LINT on the result of network monitoring queries; and (iii) evaluation of LINT-flow, including studying the impact of limiting the memory for tracking flows.

Setup

Topology and Workload We used a 4-port fat-tree data center network topology (20 switches, 32 links) for our evaluation. Each top-of-the-rack switch in each pod was connected with a traffic generating host. We enabled jumbo Ethernet frames on all the interfaces to avoid packet fragmentation during our experiments. For the workload, we used the packet capture from UNIV1 trace [214]. We divided and distributed the capture files to the Mininet hosts, and replayed them using the *tcpreplay* tool. We used ONOS controller for path setup, and for configuring the bmv2 switches to embed SwitchID, hop latency and queue occupancy on all packets. Out of the 4 pods in the topology, the hosts from pod 0 and pod 3 sent traffic to the hosts in pod 1 and pod 2, creating an aggregation traffic pattern (similar to partition–aggregate or reduce workload). This pod-to-pod path consists of 4 INT hops.

Network Monitoring Queries We used the INT reports for answering the following questions about the network:

- **(Q_{Tail}) Tail latency [95]**: Which flows have at least one packet with total hop latency in the tail latency zone? For our experiment, we use the 95-th percentile of total hop latency from all collected INT reports as the threshold for tail latency zone.
- **($Q_{Congestion}$) Congested switch identification**: Which flows have a congested switch on their path? We define a congested switch to be a switch where a packet is experiencing more than $x\%$ of its path's total hop latency. We set this threshold to 40% in our experiments.

- **($Q_{Latency}$) Path latency**: What is the total hop-latency experienced by each of the packets?
- **(Q_{Queue}) Queue profiling [95]**: Obtain the time series of queue occupancy in a switch within a given time window.

Parameter Selection We experimented with different values of α (in the form 2^{-m}) and consistently obtained the best results for $\alpha = 2^{-1}$, hence, used this value for reporting all the results. We varied δ between 2^{-6} and 2^{-1} in multiples of 2 in the experiments.

Evaluation Metrics

Recall We evaluate queries Q_{Tail} and $Q_{Congestion}$ using recall, *i.e.*, the fraction of identified flows in the tail latency zone or having a congested switch, respectively, that are also identified by the ground truth. For these two queries, we are interested in measuring the fraction of the culprit flows that can still be identified by LINT, hence, the choice of using recall.

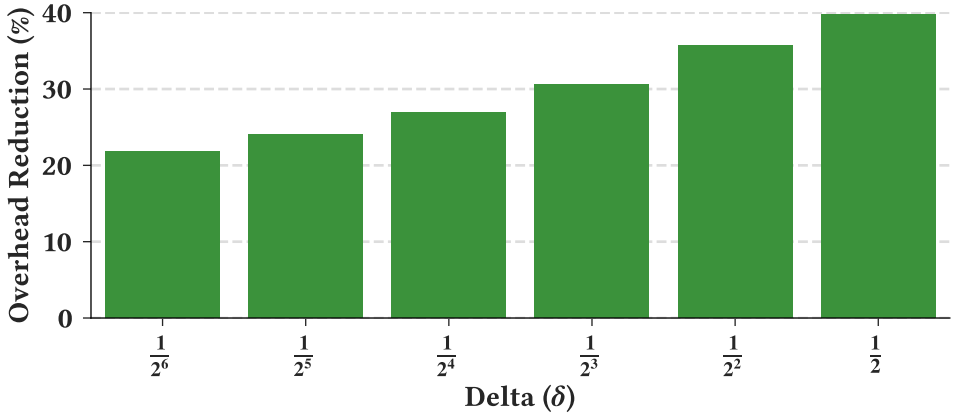
Normalized Root Mean Squared Error (NRMSE) We evaluate $Q_{Latency}$ and Q_{Queue} by measuring the deviation from the ground truth using NRMSE. For the metrics of interest in these queries, we first compute the square-root of the mean squared error (RMSE) across all the collected INT reports. Then we normalize the RMSE by the range of values of that metric and express as percentage.

Overhead reduction We measure overhead reduction by taking the ratio of the number of observations for telemetry data items that were not reported by LINT (and LINT-flow) to the total number of telemetry data item observations collected in the ground truth.

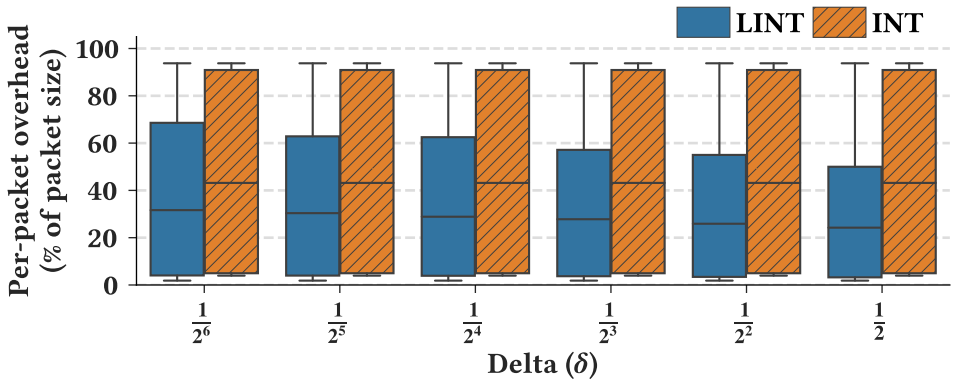
Per-packet overhead Per-packet overhead is computed as the percent increase in packet size due to embedding telemetry data items.

Data plane Overhead Reduction

Our first set of results demonstrate the effectiveness of LINT in reducing data plane overhead compared to regular INT. In Figure 5.10(a), we present the percentage overhead reduction by LINT compared to regular INT for different values of δ. The parameter δ provides a tuning



(a) Overhead reduction compared to per-packet INT



(b) Distribution of per-packet overhead

Figure 5.10: Overhead comparison between LINT and INT

knob in our algorithm to increase or decrease overhead while having an opposite effect on how often telemetry data items are reported from the data plane. As we can see, even with a very small δ ($= 2^{-6}$), LINT can reduce 20% data plane overhead compared to regular INT. We can clearly see that a higher δ also increases the gain in overhead reduction. However, this savings in overhead comes at the cost of degrading the quality of results of the monitoring queries as we will discuss in Section 5.4.3.

We also present the distribution of per-packet overhead incurred by both LINT and INT in Figure 5.10(b). The boxes in this figure represent the quartiles of the distribution while the

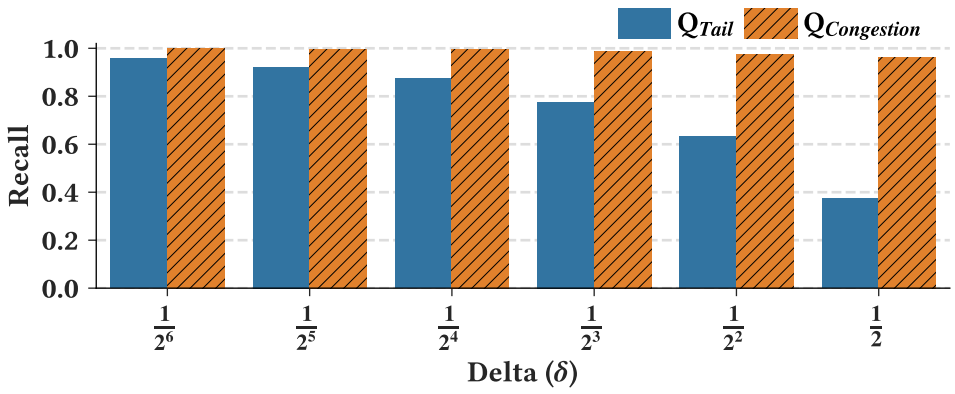


Figure 5.11: Q_{Tail} and $Q_{Congestion}$ Recall

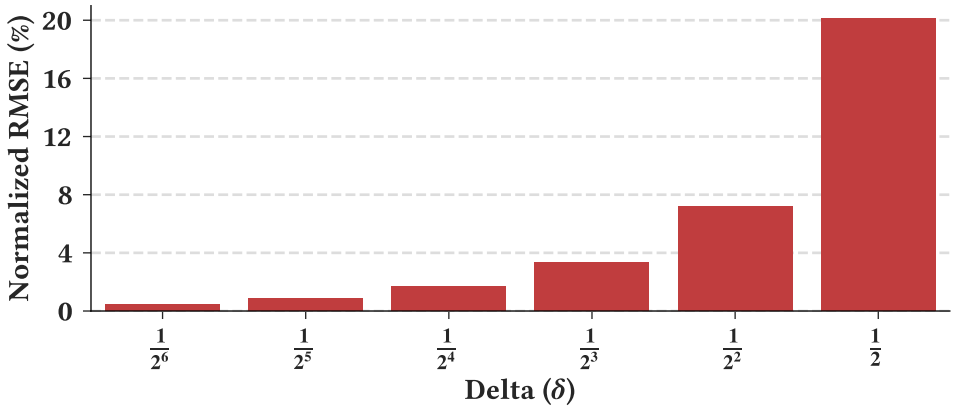
bars' endpoints represent the extremes of the distribution. Although LINT's overhead in the extreme can be as bad as INT, however, the higher quartiles are significantly smaller for LINT. For instance, for $\delta = 2^{-4}$, LINT reduces the 75th-percentile overhead of INT from ≈90% to ≈60%.

Query Performance

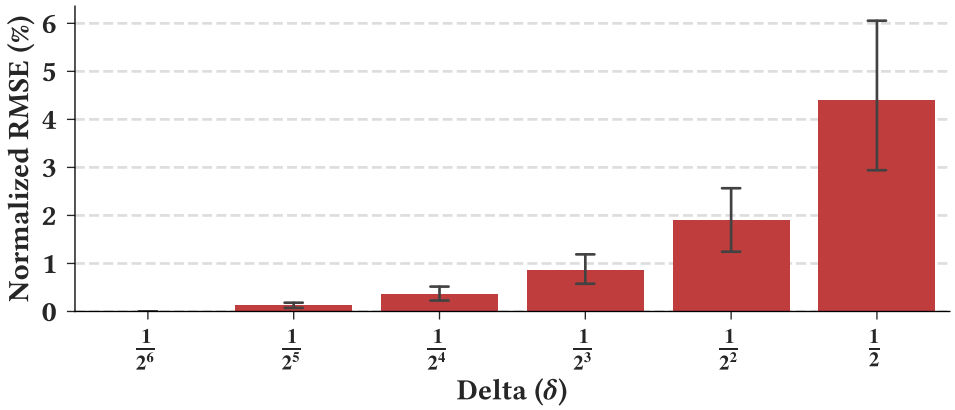
We demonstrate LINT's capability of retaining useful information even after deciding not to report some observations of the telemetry data items. We evaluate Q_{Tail} and $Q_{Congestion}$ using recall and $Q_{Latency}$ and Q_{Queue} using NRMSE.

Q_{Tail} and $Q_{Congestion}$ In Figure 5.11, we present the recall of queries Q_{Tail} and $Q_{Congestion}$ computed using the telemetry data obtained through LINT and compared against the ground truth. As noted earlier, δ is a tuning knob to find a trade-off between overhead and accuracy, which is also evident in this figure. Increasing δ causes recall of both of the queries to degrade. For Q_{Tail}, the recall starts to degrade slowly and then falls sharply. This is because errors due to selectively reporting telemetry data items causes the tail latency threshold to diverge further from that computed using the ground truth.

However, $Q_{Congestion}$ can tolerate more noise in the data as long as the ratios of latency values collected from different switches remain similar. As a result, even after estimating some of the missing values with EWMA, $Q_{Congestion}$ retains a very high recall. For instance, for the highest δ used in our experiments ($\delta = 2^{-1}$) $Q_{Congestion}$'s recall is still more than 0.95, only a few points



(a) NRMSE of total hop latency



(b) Mean NRMSE of queue occupancy for all switches

Figure 5.12: $Q_{Latency}$ and Q_{Queue} NRMSE

below its best recall (for $\delta = 2^{-6}$). However, the precision of $Q_{Congestion}$ degrades with higher δ (not shown), increasing the chances of raising false alarms. Comparing between the quality of the query results and the overhead reduction, we find $\delta = 2^{-5}$ to be a good trade-off in our experiment setting. With $\delta = 2^{-5}$, we still have a substantial overhead reduction of about 25% while maintaining a recall above 0.9 for both of the queries.

$Q_{Latency}$ and Q_{Queue} We present the NRMSE of the results of $Q_{Latency}$ and Q_{Queue} in Figure 5.12. For $Q_{Latency}$, we compute the NRMSE of the total hop latency for all INT reports collected through LINT (Figure 5.12(a)). For Q_{Queue}, we compute the NRMSE of the stream of queue occupancy observations for each of the switches collected through LINT and present the mean along with standard deviation across the switches in Figure 5.12(b). We observe a similar trend as the previous queries, *i.e.*, a higher δ degrades the quality of the query results. However, for the previously identified operating point, $\delta = 2^{-5}$, the NRMSE is minute for both $Q_{Latency}$(less than 2%) and Q_{Queue}(less than 0.25%).

LINT-flow Performance and Trade-offs

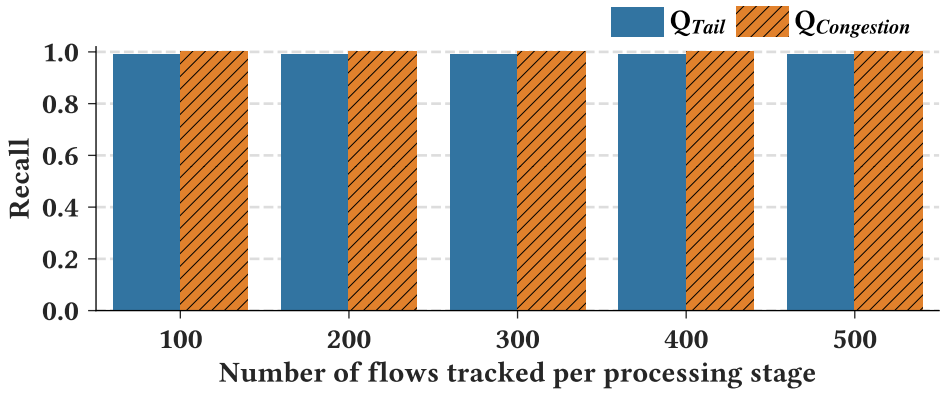


Figure 5.13: Impact of number of track flows per-stage on Q_{Tail} and $Q_{Congestion}$

We demonstrate the impact of having limited memory on LINT-flow by fixing the number of simultaneously tracked flows per processing stage (between 100 and 500) and employing least recently used (LRU) eviction policy for the old flows. We present the results on Q_{Tail} and $Q_{Congestion}$ recall in Figure 5.13. For these results we set δ to 2^{-5}. Our first observation is that considering flow-context while selectively reporting telemetry data items substantially improves the recall for both queries. Although not shown here for space constraints, the same holds for higher δ as well. However, the number of flows that can be simultaneously tracked at each stage had very little impact on the recall. We also observed similar behavior for overhead reduction. This behavior can be attributed to a combination of factors such as the short-lived nature of the flows, the use of LRU policy to exclude the old flows, and the reduction in IP address entropy due to rewriting the IP addresses in the trace with the ones of the Mininet hosts in the network. However, we plan to investigate further to identify the root cause.

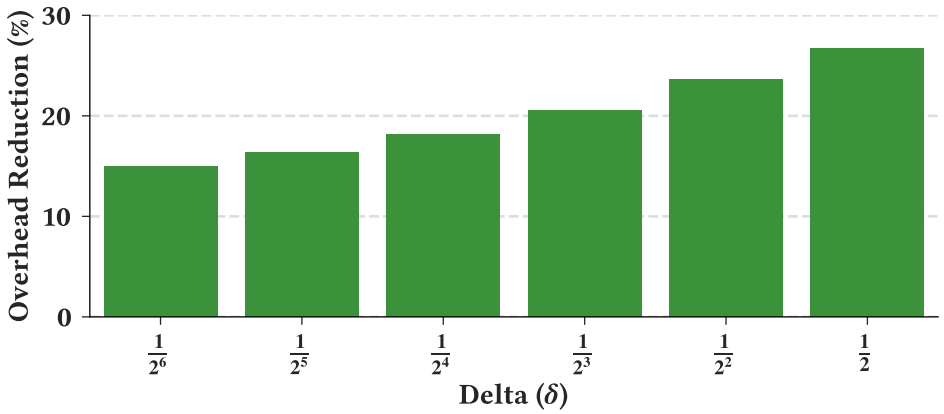


Figure 5.14: Overhead reduction by LINT-flow

Even for LINT-flow the overhead reduction is dominated by the δ parameter. We present results on overhead reduction in Figure 5.14 by fixing the number of flows simultaneously tracked at each stage to 100 and varying δ. Indeed, considering flow-context leads to reporting more telemetry data items for keeping the per-flow error estimate within bounds, consequently, reducing the gain. However, the overhead reduction still remains within 15% −25% range.

5.5 Related Works

Traditional IP Network Monitoring

There exists a number of flow based network monitoring tools for traditional IP networks. NetFlow [78] from Cisco is the most prevalent one. NetFlow probes are attached to a switch as special modules. These probes collect either complete or sampled traffic statistics, and send them to a central collector [80]. NetFlow version 9 has been adopted to be a common and universal standard by IP Flow Information Export (IPFIX) IETF working group, so that non-Cisco devices can send data to NetFlow collectors. NetFlow provides information such as source and destination IP address, port number, byte count, *etc.* It supports different technologies like multi-cast, IPSec, and MPLS. Another flow sampling method is sFlow [229], which was introduced and maintained by InMon as an open standard. It uses time-based sampling for capturing traffic information. Another proprietary flow sampling method is JFlow [79], developed by the Juniper Networks. JFlow is quite similar to NetFlow. JFlow provides detailed information about

each flow by applying statistical sampling just like NetFlow and sFlow. Except for sFlow, NetFlow and JFlow are both proprietary solutions and incur a large up-front licensing and setup cost to be deployed in a network. sFlow is less expensive to deploy, but it is not widely adopted by the vendors.

Softwarized Network Monitoring

The flow table programming capability of OpenFlow has stimulated a substantial body of research in OpenFlow network monitoring [72, 208, 230, 231]. Thanks to the flow table programming capabilities of OpenFlow, a wide-range of monitoring tasks now can be carried out only using commodity OpenFlow switches, including, traffic matrix computation [86, 232, 233], heavy-hitter or elephant flow detection [84, 234, 235, 236], latency and packet loss monitoring [237, 238, 239, 240] and network security monitoring [241, 242, 243, 244], among others. The flow table programming capabilities of OpenFlow was further extended by the recent advances in PISA network devices and the P4 programming language. PISA devices and P4 further extended network programmability, enabling programmable packet parsing and actions, and stateful operation in the data plane through the use of general purpose registers. Consequently, a substantial body of work leveraged programmable data plane capabilities for monitoring, especially for sketch-based traffic summary computation in the data plane [245]. Sketches are compact data structures with a fixed memory that can compute an approximate summary of a data stream [246]. Research in sketch-based network monitoring has been focused on designing and implementing monitoring task specific sketches (*e.g.*, for heavy-hitter detection [247], network anomaly detection [248]) for programmable data planes [249, 250]. In contrast to these task specific approaches, UnivMon proposes a universal sketch data structure that is capable of computing a wide-range of function over stream of packets [251]. Unlike INT, sketch-based approaches provide a summary computed from the network traffic instead of providing per-packet microscopic visibility into the network.

More recently, INT has emerged as a standard for providing unprecedented visibility into the network that was not possible with traditional network monitoring technologies. Since the release of initial specification, many applications of INT for network operations and management have been proposed, including for failure detection [252, 253], congestion control [96] and tracking the data plane rules matched by the flows [95, 254], among others. Although INT was initially proposed for IP networks, several extensions of INT have been proposed such as for wireless networks [255] and multi-layer IP-over-Optical networks [256, 257, 258]. It is worth mentioning that INT is one of several concurrent efforts towards performing in-band network telemetry using live network traffic (*cf.* In-situ Operations, Administration, and Maintenance (IOAM) standardization effort within the IETF [259]).

Accuracy – overhead Trade-off in SDN (OpenFlow) Monitoring

A longstanding problem in network monitoring has been addressing the trade-off between the overhead of collecting network monitoring data and the accuracy of the network view constructed from the data. For OpenFlow networks, the overhead stems from multiple sources, namely, the control plane resource usage for statistics collection and the load on the controller [208], and the data plane resource usage, *i.e.*, the usage of limited TCAM memory [89]. In this chapter, we focus on reducing the control plane overhead. A substantial body of work addressing the data plane resource usage in OpenFlow network monitoring exists in the literature. Interested readers are referred to some of the notable works in the area presented in [84, 234].

PayLess is inspired by the history of using adaptive sampling techniques in wireless sensor networks [260, 261, 262, 263, 264]. The main focus of these works have been effectively collecting sensor data while minimizing the use of their constrained resources such as battery and network bandwidth. Adaptive sampling techniques have also been used in the context of traditional IP networks [265, 266]. However, to the best of our knowledge PayLess is among one of the firsts to propose a variable rate adaptive sampling mechanism for reducing control plane overhead of OpenFlow network monitoring. Several other contemporary works such as FlowSense [209] and OpenNetMon [267] have also proposed mechanisms for reducing control plane overhead of SDN monitoring. FlowSense proposed a passive monitoring approach that relied on the information piggybacked into OpenFlow PacketIn and FlowRemoved messages. In contrast, OpenNetMon proposed a similar adaptive sampling approach for querying the switches from the control plane.

A substantial number of works in the research literature have since addressed the control plane overhead issue in OpenFlow network monitoring [72, 208]. For instance, OpenSample, one of the early works in SDN monitoring, proposes to leverage the sFlow functionality available in switches to construct a global network view at the control plane [268]. Instead of relying on OpenFlow counters, OpenSample configures the switch ports to send sampled packets to the control plane. Sampled packets allow the control plane to compute approximate statistics about the network flows, however, the flow counters in the OpenFlow switches provide an accurate statistics of that particular flow. PayLess and many of the subsequent works in the literature rely on the flow counters as opposed to on sampled packets. Apart from adjusting the monitoring frequency, other works in the literature have also focused on aspects such as adapting the set of switches and the set of flows to monitor [269, 270, 271], and adapting the granularity of flow rules to monitor [84, 272, 273] for reducing SDN monitoring overhead. For a detailed survey on these approaches and others, the readers are referred to some of the field surveys available in [72, 208, 231].

INT Overhead Reduction

One approach for reducing INT overhead involves using probe packets to collect telemetry data from INT capable devices instead of piggybacking the same on live network traffic [274, 275]. Probing the network in this way requires carefully crafting the probe packets and planning the probe paths for maximum network coverage. Pan *et al.*, addresses this problem by proposing an optimization based approach in [274]. However, probe packets are often not subjected to the same treatment as the live network traffic, therefore, can obtain an incorrect view of the network.

Several approaches have been proposed to reduce INT overhead for live network traffic. For instance, Marques *et al.*, have proposed an offline optimization approach for INT in [276]. Their approach assumes the knowledge of all network flows and devises an offline schedule for what telemetry data item should be collected by packets of which flows while considering constraints such as MTU limitations. An online approach for reducing INT overhead is presented by Tang *et al.*, in [101]. They implement INT capabilities in Open vSwitch and employ sampling for deciding which packets should be subjected to INT along the way. A central controller adjusts the sampling rate at the end hosts and configures a watchlist of flows to monitor. In contrast to these aforementioned approaches, we propose an online mechanism that works completely in the data plane without the intervention of a centralized controller. Also, each switch independently decides on which telemetry data items to report without any global coordination. In contrast to the sampling-based approaches such as those presented in [101, 252], we propose to adapt telemetry data reporting based on error estimates computed within the data plane.

Very recently PINT [102] proposed a randomized algorithm for INT. PINT fixes the bit overhead allowed on a packet for INT. Then, each network device makes a random decision for embedding INT data. Since switches randomly decide on embedding INT data, therefore, the requested telemetry data items can be reported across multiple packets. PINT also proposes mechanisms for minimizing the number of packets required to collect all required telemetry data items. PINT is effective for executing network monitoring queries that work with aggregate data and when network flows are not short-lived. In contrast to PINT, we propose a complimentary approach for supporting network monitoring queries that rely on per-hop telemetry data and is oblivious to flow duration.

5.6 Chapter Summary

In this chapter, we have addressed the issue of striking a balance between network monitoring overhead and accuracy considering both control and data plane overhead. In this context, we

first introduced PayLess, a traffic-intensity adaptive variable frequency algorithm for monitoring an SDN from the control plane. We implemented a network link utilization monitoring use-case using the PayLess algorithm on Floodlight OpenFlow controller. We have evaluated and compared its performance with that of Flowsense, a passive zero cost link utilization monitoring algorithm for OpenFlow networks, and a periodic polling method. We found that PayLess can achieve higher accuracy of statistics collection than Flowsense. Yet, the incurred messaging overhead is up to 50% of the overhead in an equivalent periodic poling strategy.

We also presented LINT, an accuracy-adaptive and lightweight INT mechanism. LINT operates entirely in the data plane without any control plane intervention and without any global co-ordination. We also proposed LINT-flow, an extension of LINT that takes each packet's flow-context into consideration for selectively reporting telemetry data. We evaluated LINT using a real data-center traffic trace. Our evaluation results demonstrate the effectiveness of LINT in reducing data plane overhead by $\approx$25% while maintaining more than 0.9 recall for network monitoring queries trying to identify flows with high latency and flows with congested switches in the network. Furthermore, with appropriate parameter settings switch queue occupancy and switch processing delay computed from telemetry data collected using LINT exhibited less than 0.25% and less than 2% normalized RMSE, respectively. Even across all parameter values used in the experiments, LINT achieved comparable data plane overhead reduction while incurring $\approx$5% NRMSE on average.

Chapter 6

Conclusion and Future Work

6.1 Conclusion

Telecommunications and data center network operators are increasingly adopting network softwarization for diversifying their supply chain, enabling on-demand service provisioning, achieving better control over the network resources, and accelerating the time-to-market for new services. However, reaping the full benefits of softwarization requires resource management mechanisms that can take full advantage of the flexibility brought forth by softwarization. In this context, this dissertation challenged some of the current resource management practices and addressed the shortcomings in how resource allocation is performed and how softwarized networks are monitored. Specifically, we addressed four resource management challenges in three key enablers of network softwarization, namely SDN, NFV, and network virtualization. In the following, we summarize our contributions.

In Chapter 2, we presented the design and implementation of μNF, a system for building VNFs and SFCs from reusable, independently deployable, and loosely-coupled components enabling finer-grained resource allocation and scaling. Our design goal has been to keep the μNFs simple and develop the necessary primitives to transparently enable different communication patterns between them. We demonstrated the effectiveness of our system through a DPDK based prototype implementation and experimental evaluation. We employed a number of techniques such as batched I/O, zero-copy kernel-bypass network I/O, parallelization and cache pre-fetching for implementing and optimizing the system. The main takeaway is our demonstration that disaggregating complex VNFs using the proposed software architecture combined with the above techniques is indeed a viable and competitive solution for composing VNFs and SFCs. This is further supported by our experimental evaluation showing that the com-

bined engineering effort enables finer-grained resource allocation and scaling while attaining comparable performance as state-of-the-art monolithic implementations. Key results from our testbed evaluation are: (i) compared to monolithic VNF based SFCs, μNF-based ones achieve the same throughput by using less CPU cycles per packet on average (≈17% less CPU cycles in our experiment setup); (ii) compared to NetBricks [1], state-of-the-art system for realizing SFCs in a run-to-completion manner, μNF-based SFCs require lesser number of CPU cores for achieving the same packet processing throughput.

In Chapter 3, we studied the **MUL**ti-layer virtual network **E**mbedding (MULE) problem that embeds a VN on a multi-layer IP-over-OTN, and jointly optimizes the cost of allocating resources for the VN and the cost of creating new IP links as necessary. We have proposed an ILP formulation, OPT-MULE, for optimally solving MULE. To the best of our knowledge, this is the first optimal solution to the multi-layer VNE problem. We have also devised a polynomial time heuristic, FAST-MULE, to address the computational complexity of the optimal solution. FAST-MULE collapses the multi-layer network into a single-layer network and reduces the joint computation of virtual node and link embedding, and creation of new IP links and their mapping to an instance of computing maximum flow in the collapsed graph. We have shown that our proposed polynomial time heuristic obtains optimal solution for a special class of VNs, namely, for star shaped VNs with uniform bandwidth demand. Our empirical evaluation of FAST-MULE shows that it performs within 1.47× of the optimal solution on average. FAST-MULE also outperformed state-of-the-art heuristic for the multi-layer VNE problem and allocated ≈66% less resources while accepting ≈60% more VN requests on average.

In Chapter 4, we studied the 1 + 1 – **Pro**tected **Vi**rtual **N**twork **E**mbedding (1 + 1 – ProViNE) problem that embeds a VN on an SN while ensuring dedicated backup for each virtual node and link, mutually exclusive from their primary embedding. To this end, we have presented DRONE, a suite of solutions for 1 + 1 – ProViNE. We devised an ILP based optimal solution (OPT-DRONE) for DRONE, improving over the quadratic linear program solution in state-of-the-art [3]. We also proposed a polynomial time heuristic algorithm (FAST-DRONE) to tackle the computational complexity of the optimal solution. Our heuristic is constructed on a restructured representation of DRONE where we transform DRONE into an instance of jointly performing partitioning of the SN into two parts and VNE within the two partitions. We have evaluated our solutions using both real and synthetic network topologies and under both static and dynamic scenarios. Our evaluation results show that FAST-DRONE can solve 1 + 1 – ProViNE in a reasonable time frame allocating only 14.3% additional resources on average compared to OPT-DRONE. Our evaluation results also show that FAST-DRONE can accept 4× more VN requests compared to state-of-the-art heuristic [3].

Finally, in Chapter 5, we have addressed a longstanding issue in network monitoring, namely, striking a balance between network monitoring overhead and the accuracy of the network view

constructed from monitoring data in the context of softwarized networks. In this context, we first introduced PayLess, a traffic-intensity adaptive variable frequency algorithm for monitoring an SDN from the control plane. PayLess focuses on reducing the *control plane overhead* of softwarized network monitoring without substantially sacrificing the accuracy. We have implemented a network link utilization monitoring use-case using PayLess on Floodlight OpenFlow controller. We have evaluated and compared PayLess with FlowSense, a passive link utilization monitoring algorithm for OpenFlow networks, and a periodic polling method as a baseline. We found that PayLess can achieve higher accuracy of statistics collection than FlowSense. Yet, the incurred messaging overhead is up to 50% of the overhead in an equivalent periodic poling strategy. In the same vein, we also present LINT, an accuracy-adaptive and lightweight INT mechanism. In contrast to PayLess, LINT focuses on reducing the *data plane overhead* of network monitoring. LINT operates entirely in the data plane without any control plane intervention and without any global co-ordination. We also propose LINT-flow, a variation of LINT that takes each packet's flow-context into consideration for selectively reporting telemetry data. We evaluate LINT using a real data-center traffic trace. Our evaluation results demonstrate the effectiveness of LINT in reducing data plane overhead by 25% while maintaining more than 0.9 recall for network monitoring queries trying to identify flows with high latency and flows with congested switches in the network. Furthermore, with appropriate parameter settings switch queue occupancy and switch processing delay computed from telemetry data collected using LINT exhibited less than 0.25% and less than 2% normalized RMSE, respectively. Even across all parameter values used in the experiments, LINT achieved comparable data plane overhead reduction while incurring $\approx$5% NRMSE on average.

In this dissertation, we have addressed four key resource management challenges, paving the way for effective resource management in next generation communication networks. We have employed various methodologies, including, system design and implementation, and optimization and algorithm design for proposing novel solutions to these problems. We have also demonstrated the effectiveness and superiority of our proposed solutions through extensive testbed evaluation, network emulation and simulations using realistic scenarios, and comparing to state-of-the-art solutions when possible. In the following, we briefly discuss some interesting future research directions and some issues that require further investigation.

6.2 Future Research Direction

6.2.1 VNF Disaggregation

Granularity of VNF Decomposition In this thesis, we proposed a system for composing SFCs and VNFs from independently deployable loosely-coupled μNFs. Orthogonal to the system design is the identification of the set of μNFs in the first place. From our initial survey this appears to be rather challenging primarily because it requires domain specific knowledge. Also determining the granularity of such tasks is non-trivial. On one hand, most of the academic works propose low level packet processing functions (*e.g.*, TCP processing functions [127]) as VNF building blocks. On the other hand, state-of-the-art commercial VNFs [152] are composed from coarser-grained building blocks. Finer granularity increases re-usability whereas coarser granularity reduces overhead. *The best way to decompose a VNF into μNFs* remains an interesting research question.

Packet Ownership Transfer When a μNF is finished processing a packet and transfers it to another μNF, the ownership of the packet should be transferred to that other μNF as well, *i.e.*, the previous μNF should not be able to access the packet content using the previously acquired packet handler. Virtual switches provide this abstraction by copying packets between ports, so, the previous copy becomes invalidated. However, this is a difficult problem to solve using a shared memory subsystem. In our implementation, μNFs rely on the `hugetlbfs` to obtain virtual-to-physical memory translation of the packet addresses. This file system should be accessible to the μNFs to ensure that they can always obtain a valid translation. This requirement also raises the issue that μNFs can always read packet content even after the packet has been transferred to other μNFs, and consequently, ownership is not transferred. Ownership transfer between multiple processes has been studied in HPC systems [277]. However, the state-of-the-art in that area still performs at least one message copy, which in our case would add a significant latency in packet processing. *Ownership transfer in shared-memory multi-process system with zero-copy* remains an open question. As a workaround in our implementation we created disjoint segments in the huge table area and assign one area to μNFs of the same processing graph. This does not solve the problem 100%, however, it provides isolation between μNFs from different processing graphs.

μNF Orchestration Our contribution in this dissertation primarily focused on developing a working solution and addressing the engineering challenges for enabling VNF and SFC composition from independently deployable μNFs while operating at line rate. However, to get

the best out of such architecture we also need to address orchestration problems such as μNF graph optimization, incorporating parallelization and consolidation of μNF instances, optimal placement of μNF graph across multiple machines, scaling out μNF instances across multiple machines, state management between scaled out instances, fault-tolerance, and scheduling of μNF instances for better resource utilization, among others.

6.2.2 Transport SDN Virtualization

Multi-layer VNE In this dissertation, we have considered a deployment scenario where an IP network is provisioned on top of a static OTN. One immediate future direction can be considering a dynamic OTN where more capacity can be provisioned by establishing new light paths in the underlying DWDM network. In this case, one needs to take technological constraints of DWDM networks into account such as wavelength continuity constraints. These additional physical layer constraints add new dimension to the problem and merits a separate investigation. Another emerging technology for deploying the physical layer of transport networks is Elastic Optical Networks (EONs) [278]. In contrast to DWDM networks, EONs allow finer-grained spectrum resource allocation capabilities and the flexibility to tune transmission parameters such as modulation format, forward error correction overhead, and baud-rate [279]. At the same time, EONs bring additional physical layer constraints such as ensuring both the continuity and contiguity of spectrum allocation on a light path. Therefore, solving the multi-layer VNE problem for IP-over-EON multi-layer network comes with its unique challenges and is an interesting future research direction.

Survivable VNE We believe that the formulation of the 1 + 1 – ProViNE problem will open up new avenues for future research. One such possibility is to investigate the problem of providing mixed backup scheme for the VNs. A mixed backup scheme consists of providing both shared and dedicated backup to the VN elements based on the service provider's request. A mixed backup scheme can enable the service providers to have dedicated protection for critical network paths while shared protection for best effort network paths for instance. Another interesting research direction is to investigate how exactly traffic will be switched from the primary to the backup embedding when such dedicated protection is present. In this context, ensuring the consistency of network state (*e.g.*, routing table, access control lists, rate limiters) between the primary and the backup instances of the VN elements is of practical importance and requires further investigation. Furthermore, dedicated protection for VNs can also be considered for emerging EON enabled transport networks, which brings its unique challenges and merits further investigation.

6.2.3 Softwarized Network Monitoring

Adaptive monitoring from the control plane In PayLess, we resorted to using static thresholds for traffic intensity change while deciding on increasing or decreasing the polling frequency. An issue with using static threshold is that the threshold has to be set and tuned based on domain expertise. An interesting future research direction is to devise mechanisms for automatically setting and adjusting the thresholds based on the characteristics of network traffic. In this regard, the use of Machine Learning (ML) techniques can be a promising direction to pursue. Furthermore, the use of ML techniques can be leveraged to determine the polling frequency itself. Another aspect of adaptive monitoring from the control plane is to also determine the set of switches and the flows on those switches to be monitored. Although, this was not the focus of PayLess, a substantial body of literature exists in this area. However, there is still room in the literature for exploring the use of ML techniques in this context. An interesting direction along this line would be to explore the use of Graph Neural Networks that capture the network topology and the traffic flow for automatically adapt the set of switches to monitor, the set of flows in these switches to monitor, and the frequency of polling statistics about those flows.

Adaptive monitoring from the data plane While designing and implementing LINT, we resorted to using EWMA function for predicting the missing values and estimating error in the data plane. Indeed, we have seen promising results from using EWMA. However, an interesting avenue for further investigation is to explore what other predictor functions can be used in the data plane and evaluate the trade-offs between their computation complexity and quality of the results of monitoring queries. If some predictor functions cannot be directly implemented in the data plane due to the data plane constraints, one would need to resort to approximation techniques such as fixed-point arithmetic and using pre-computed tables in the data plane. Doing so would also create another trade-off between data plane resource usage and the accuracy of approximation, which merits a separate investigation. Another interesting research direction to pursue is to investigate the use of information theory measures for quality of information aware adaptive telemetry.

References

[1] A. Panda, S. Han, K. Jang, M. Walls, S. Ratnasamy, and S. Shenker, "Netbricks: Taking the V out of NFV," in *Proceedings of USENIX OSDI*, 2016, pp. 203–216.

[2] J. Zhang, Y. Ji, M. Song, H. Li, R. Gu, Y. Zhao, and J. Zhang, "Dynamic virtual network embedding over multilayer optical networks," *IEEE/OSA Journal of Optical Communications and Networking*, vol. 7, no. 9, pp. 918–927, September 2015.

[3] Z. Ye, A. N. Patel, P. N. Ji, and C. Qiao, "Survivable virtual infrastructure mapping with dedicated protection in transport software-defined networks [invited]," *IEEE/OSA Journal of Optical Communications and Networking*, vol. 7, no. 2, pp. A183–A189, February 2015.

[4] "IMT vision – framework and overall objectives of the future development of IMT for 2020 and beyond," Recommendation ITU-R M.2083-0, 2015.

[5] 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, October 2016.

[6] A. Feldmann, O. Gasser, F. Lichtblau, E. Pujol, I. Poese, C. Dietzel, D. Wagner, M. Wichtlhuber, J. Tapiador, N. Vallina-Rodriguez, O. Hohlfeld, and G. Smaragdakis, "The lockdown effect: Implications of the covid-19 pandemic on internet traffic," in *Proceedings of the ACM Internet Measurement Conference*, ser. IMC '20, 2020, p. 1–18.

[7] A. Lutu, D. Perino, M. Bagnulo, E. Frias-Martinez, and J. Khangosstar, "A characterization of the covid-19 pandemic impact on a mobile network operator traffic," in *Proceedings of the ACM Internet Measurement Conference*, ser. IMC '20, 2020, p. 19–33.

[8] "Cisco visual networking index: Forecast and trends, 2018–2023," White paper, Cisco Systems. [Online]. Available: https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html

[9] "Network slicing use case requirements," Reference Document, GSMA Association, June 2018. [Online]. Available: https://www.gsma.com/futurenetworks/wp-content/uploads/2018/06/Network-Slicing-Use-Case-Requirements-_-FInal-.pdf

[10] X. Foukas, G. Patounas, A. Elmokashfi, and M. K. Marina, "Network slicing in 5G: Survey and challenges," *IEEE Communications Magazine*, vol. 55, no. 5, pp. 94–100, 2017.

[11] I. Parvez, A. Rahmati, I. Guvenc, A. I. Sarwat, and H. Dai, "A survey on low latency towards 5G: Ran, core network and caching solutions," *IEEE Communications Surveys & Tutorials*, vol. 20, no. 4, pp. 3098–3130, Fourth quarter 2018.

[12] D. G. Messerschmitt, "The convergence of telecommunications and computing: what are the implications today?" *Proceedings of the IEEE*, vol. 84, no. 8, pp. 1167–1186, August 1996.

[13] T. V. Doan, V. Bajpai, and S. Crawford, "A longitudinal view of netflix: Content delivery over ipv6 and content cache deployments," in *Proceedings of IEEE INFOCOM*, 2020, pp. 1073–1082.

[14] C. Perera, Y. Qin, J. C. Estrella, S. Reiff-Marganiec, and A. V. Vasilakos, "Fog computing for sustainable smart cities: A survey," *ACM Computing Surveys*, vol. 50, no. 3, Jun. 2017.

[15] M. Calder, X. Fan, Z. Hu, E. Katz-Bassett, J. Heidemann, and R. Govindan, "Mapping the expansion of google's serving infrastructure," in *Proceedings of ACM Internet Measurement Conference*, 2013, p. 313–326.

[16] M. Ammar, E. Zegura, and Y. Zhao, "A vision for zero-hop networking (zen)," in *Proceedings of IEEE ICDCS*, 2017, pp. 1765–1770.

[17] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu, J. Zolla, U. Hölzle, S. Stuart, and A. Vahdat, "B4: Experience with a globally-deployed software defined wan," in *Proceedings of ACM SIGCOMM Conference*, 2013, pp. 3–14.

[18] C.-Y. Hong, S. Kandula, R. Mahajan, M. Zhang, V. Gill, M. Nanduri, and R. Wattenhofer, "Achieving high utilization with software-driven wan," in *Proceedings of ACM SIGCOMM Conference*, 2013, pp. 15–26.

[19] K.-K. Yap, M. Motiwala, J. Rahe, S. Padgett, M. Holliman, G. Baldus, M. Hines, T. Kim, A. Narayanan, A. Jain, V. Lin, C. Rice, B. Rogan, A. Singh, B. Tanaka, M. Verma, P. Sood,

M. Tariq, M. Tierney, D. Trumic, V. Valancius, C. Ying, M. Kallahalla, B. Koley, and A. Vahdat, "Taking the edge off with espresso: Scale, reliability and programmability for global internet peering," in *Proceedings of ACM SIGCOMM Conference*, 2017, pp. 432–445.

[20] B. Schlinker, H. Kim, T. Cui, E. Katz-Bassett, H. V. Madhyastha, I. Cunha, J. Quinn, S. Hasan, P. Lapukhov, and H. Zeng, "Engineering egress with edge fabric: Steering oceans of content to the world," in *Proceedings of ACM SIGCOMM Conference*, 2017, pp. 418–431.

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

[22] M. Casado, M. J. Freedman, J. Pettit, J. Luo, N. McKeown, and S. Shenker, "Ethane: Taking control of the enterprise," in *Proceedings of ACM SIGCOMM Conference*, 2007, pp. 1–12.

[23] 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 Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, March 2008.

[24] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, and D. Walker, "P4: Programming protocol-independent packet processors," *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 3, p. 87–95, July 2014.

[25] B. Carpenter and S. Brim, "Middleboxes: Taxonomy and issues," Internet Requests for Comments, RFC Editor, RFC 3234, February 2002. [Online]. Available: http://www.rfc-editor.org/rfc/rfc3234.txt

[26] "Network Functions Virtualisation – Introductory White Paper," White paper, The European Telecommunications Standards Institute (ETSI), Oct 2012. [Online]. Available: https://portal.etsi.org/nfv/nfv_white_paper.pdf

[27] 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," in *Proceedings of ACM SIGCOMM Conference*, 2012, pp. 13–24.

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

[29] J. S. Turner and D. E. Taylor, "Diversifying the internet," in *Proceedings of IEEE GLOBECOM*, 2005, pp. 755–760.

[30] P. Rost, A. Banchs, I. Berberana, M. Breitbach, M. Doll, H. Droste, C. Mannweiler, M. A. Puente, K. Samdanis, and B. Sayadi, "Mobile network architecture evolution toward 5G," *IEEE Communications Magazine*, vol. 54, no. 5, pp. 84–91, May 2016.

[31] X. Foukas, G. Patounas, A. Elmokashfi, and M. K. Marina, "Network slicing in 5G: Survey and challenges," *IEEE Communications Magazine*, vol. 55, no. 5, pp. 94–100, May 2017.

[32] A. Gupta, R. MacDavid, R. Birkner, M. Canini, N. Feamster, J. Rexford, and L. Vanbever, "An industrial-scale software defined internet exchange point," in *Proceedings of USENIX NSDI*, 2016, pp. 1–14.

[33] "Verizon sdn/nfv reference architecture," White paper, Verizon, May 2016.

[34] "Flexible network-based, enterprise-class software-defined solutions for hybrid public/private network connectivity," White paper, AT&T, June 2018. [Online]. Available: https://www.business.att.com/content/dam/attbusiness/briefs/att-sd-wan-solutions-whitepaper.pdf

[35] S. Tse and G. Choudhury, "Real-time traffic management in AT&T's sdn-enabled core ip/optical network," in *Proceedings of OSA Optical Fiber Communication Conference*, 2018, pp. Tu3H–2.

[36] M. S. Bonfim, K. L. Dias, and S. F. Fernandes, "Integrated NFV/SDN architectures: A systematic literature review," *ACM Computing Surveys*, vol. 51, no. 6, pp. 114:1–114:39, February 2019.

[37] I. Afolabi, T. Taleb, K. Samdanis, A. Ksentini, and H. Flinck, "Network slicing and softwarization: A survey on principles, enabling technologies, and solutions," *IEEE Communications Surveys & Tutorials*, vol. 20, no. 3, pp. 2429–2453, Third quarter 2018.

[38] "Management concept, architecture and requirements for mobile networks that include virtualized network functions," Technical specification 28.500 Version 16.0.0, July 2020.

[39] "Telus network as a service." [Online]. Available: https://www.telus.com/en/bc/business/data-networks/naas

[40] "AT&T switched ethernet with network on demand." [Online]. Available: https://smallbusiness.att.com/switchedethernet/

[41] "Bell virtual network service." [Online]. Available: https://business.bell.ca/shop/medium-large/internet-private-networks/virtual-network-services

[42] Y. Zhang, L. Cui, W. Wang, and Y. Zhang, "A survey on software defined networking with multiple controllers," *Springer Journal of Network and Computer Applications*, vol. 103, pp. 101 – 118, February 2018.

[43] "Open platform for NFV (OPNFV)." [Online]. Available: https://www.opnfv.org/

[44] "Open source MANO," https://osm.etsi.org/. [Online]. Available: https://osm.etsi.org/

[45] S. Palkar, C. Lan, S. Han, K. Jang, A. Panda, S. Ratnasamy, L. Rizzo, and S. Shenker, "E2: a framework for NFV applications," in *Proceedings of ACM SOSP*, 2015, pp. 121–136.

[46] P. Quinn and T. Nadeau, "Problem statement for service function chaining," Internet Requests for Comments, RFC 7498, April 2015. [Online]. Available: http://www.rfc-editor.org/rfc/rfc7498.txt

[47] V. Sekar, N. Egi, S. Ratnasamy, M. K. Reiter, and G. Shi, "Design and implementation of a consolidated middlebox architecture," in *Proceedings of USENIX NSDI*, 2012, pp. 323–336.

[48] A. Bremler-Barr, Y. Harchol, and D. Hay, "OpenBox: A software-defined framework for developing, deploying, and managing network functions," in *Proceedings of ACM SIGCOMM Conference*, 2016, pp. 511–524.

[49] S. R. Chowdhury, M. A. Salahuddin, N. Limam, and R. Boutaba, "Re-architecting nfv ecosystem with microservices: State of the art and research challenges," *IEEE Network*, vol. 33, no. 3, pp. 168–176, May 2019.

[50] R. Roseboro, "Cloud-native nfv architecture for agile service creation & scaling," White paper, January 2016.

[51] M. F. Bari, R. Boutaba, R. Esteves, L. Z. Granville, M. Podlesny, M. G. Rabbani, Q. Zhang, and M. F. Zhani, "Data center network virtualization: A survey," *IEEE Communications Surveys & Tutorials*, vol. 15, no. 2, pp. 909–928, Second quarter 2013.

[52] "Amazon Virtual Private Cloud," http://aws.amazon.com/vpc/.

[53] "SDN Architecture for Transport Networks," White paper, Open Networking Foundation (ONF), March 2016. [Online]. Available: https://www.opennetworking.org/wp-content/uploads/2014/10/SDN_Architecture_for_Transport_Networks_TR522.pdf

[54] "Global Transport SDN Prototype Demonstration," White paper, Open Networking Foundation (ONF), October 2014. [Online]. Available: https://www.opennetworking.org/images/stories/downloads/sdn-resources/technical-reports/oif-p0105_031_18.pdf

[55] N. M. K. Chowdhury, M. R. Rahman, and R. Boutaba, "Virtual network embedding with coordinated node and link mapping," in *Proceedings of IEEE INFOCOM*, 2009, pp. 783–791.

[56] A. Razzaq and M. S. Rathore, "An approach towards resource efficient virtual network embedding," in *Evolving Internet (INTERNET), 2010 Second International Conference on*. IEEE, 2010, pp. 68–73.

[57] J. F. Botero, X. Hesselbach, M. Duelli, D. Schlosser, A. Fischer, and H. De Meer, "Energy efficient virtual network embedding," *IEEE Communications Letters*, vol. 16, no. 5, pp. 756–759, May 2012.

[58] S. Ayoubi, C. Assi, K. Shaban, and L. Narayanan, "MINTED: Multicast virtual network embedding in cloud data centers with delay constraints," *IEEE Transactions on Communications*, vol. 63, no. 4, pp. 1291–1305, April 2015.

[59] A. Fischer, J. F. Botero, M. T. Beck, H. De Meer, and X. Hesselbach, "Virtual network embedding: A survey," *IEEE Communications Surveys & Tutorials*, vol. 15, no. 4, pp. 1888–1906, Fourth quarter 2013.

[60] S. Ramamurthy and B. Mukherjee, "Survivable WDM mesh networks. Part I-protection," in *INFOCOM'99.*, vol. 2. IEEE, 1999, pp. 744–751.

[61] P. A. Bonenfant, "Optical layer survivability: a comprehensive approach," in *Optical Fiber Communication Conference and Exhibit, 1998. OFC'98., Technical Digest.* IEEE, 1998, pp. 270–271.

[62] X. Zhao, V. Vusirikala, B. Koley, V. Kamalov, and T. Hofmeister, "The prospect of inter-data-center optical networks," *IEEE Communications Magazine*, vol. 51, no. 9, pp. 32–38, September 2013.

[63] N. Ghani, S. Dixit, and T.-S. Wang, "On IP-over-WDM integration," *IEEE Communications Magazine*, vol. 38, no. 3, pp. 72–84, March 2000.

[64] X. Jin, Y. Li, D. Wei, S. Li, J. Gao, L. Xu, G. Li, W. Xu, and J. Rexford, "Optimizing bulk transfers with software-defined optical wan," in *Proceedings of ACM SIGCOMM Conference*, 2016, pp. 87–100.

[65] A. L. Chiu, G. Choudhury, G. Clapp, R. Doverspike, M. Feuer, J. W. Gannett, J. Jackel, G. T. Kim, J. G. Klincewicz, T. J. Kwon, G. Li, P. Magill, J. M. Simmons, R. A. Skoog, J. Strand, A. Von Lehmen, B. J. Wilson, S. L. Woodward, and D. Xu, "Architectures and protocols for capacity efficient, highly dynamic and highly resilient core networks [invited]," *IEEE/OSA Journal of Optical Communications and Networking*, vol. 4, no. 1, pp. 1–14, 2012.

[66] C. Janz, L. Ong, K. Sethuraman, and V. Shukla, "Emerging transport sdn architecture and use cases," *IEEE Communications Magazine*, vol. 54, no. 10, pp. 116–121, October 2016.

[67] "Cisco nLight™technology: A multi-layer control plane architecture for IP and optical convergence," White paper, Cisco Systems, 2012. [Online]. Available: https://www.cisco.com/c/en/us/products/collateral/switches/catalyst-3750-series-switches/whitepaper_c11-718852.pdf

[68] S. Dahlfort and D. CAVIGLIA, "IP-optical convergence: a complete solution," *Ericsson Review*, May 2014.

[69] M. R. Rahman, I. Aib, and R. Boutaba, "Survivable virtual network embedding," in *IFIP Networking 2010*. Springer, 2010, pp. 40–52.

[70] W. Wang, Y. Lin, Y. Zhao, G. Zhang, J. Zhang, J. Han, H. Chen, B. Hou, Y. Ji, and L. Zong, "First demonstration of virtual transport network services with multi-layer protection schemes over flexi-grid optical networks," *IEEE Communications Letters*, vol. 20, no. 2, pp. 260–263, 2016.

[71] T. Holterbach, E. C. Molero, M. Apostolaki, A. Dainotti, S. Vissicchio, and L. Vanbever, "Blink: Fast connectivity recovery entirely in the data plane," in *USENIX NSDI*, 2019, pp. 161–176.

[72] I. F. Akyildiz, A. Lee, P. Wang, M. Luo, and W. Chou, "A roadmap for traffic engineering in sdn-openflow networks," *Computer Networks*, vol. 71, pp. 1–30, October 2014.

[73] M. Alizadeh, T. Edsall, S. Dharmapurikar, R. Vaidyanathan, K. Chu, A. Fingerhut, V. T. Lam, F. Matus, R. Pan, N. Yadav *et al.*, "Conga: Distributed congestion-aware load balancing for datacenters," in *Proceedings of the 2014 ACM conference on SIGCOMM*, 2014, pp. 503–514.

[74] T. De Schepper, S. Latré, and J. Famaey, "Flow management and load balancing in dynamic heterogeneous lans," *IEEE Transactions on Network and Service Management*, vol. 15, no. 2, pp. 693–706, 2018.

[75] V. Sekar, N. G. Duffield, O. Spatscheck, J. E. van der Merwe, and H. Zhang, "Lads: Large-scale automated ddos detection system." in *USENIX ATC*, 2006, pp. 171–184.

[76] J. Boite, P. Nardin, F. Rebecchi, M. Bouet, and V. Conan, "Statesec: Stateful monitoring for ddos protection in software defined networks," in *IEEE NetSoft*, 2017, pp. 1–9.

[77] B. Claise and R. Wolter, *Network Management: Accounting and Performance Strategies*. Cisco Press, 2006.

[78] "Introduction to cisco ios® netflow," White paper, Cisco Systems, May 2012. [Online]. Available: https://www.cisco.com/c/en/us/products/collateral/ios-nx-os-software/ios-netflow/prod_white_paper0900aecd80406232.pdf

[79] A. C. Myers, "JFlow: Practical mostly-static information flow control," in *Proceedings of ACM SIGPLAN-SIGACT POPL*, 1999, pp. 228–241.

[80] C. Systems, "Cisco CNS NetFlow Collection Engine," http://www.cisco.com/en/US/products/sw/netmgtsw/ps1964/index.html.

[81] "sFlow sampling rate guideline." [Online]. Available: https://blog.sflow.com/2009/06/sampling-rates.html

[82] P. Phaal, S. Panchen, and N. McKee, "Inmon corporation's sflow: A method for monitoring traffic in switched and routed networks," Internet Requests for Comments, RFC Editor, RFC 3176, September 2001. [Online]. Available: http://www.rfc-editor.org/rfc/rfc3176.txt

[83] C. Estan, K. Keys, D. Moore, and G. Varghese, "Building a better netflow," *ACM SIGCOMM Computer Communication Review*, vol. 34, no. 4, pp. 245–256, 2004.

[84] M. Moshref, M. Yu, R. Govindan, and A. Vahdat, "DREAM: dynamic resource allocation for software-defined measurement," in *Proceedings of ACM SIGCOMM Conference*, 2015, pp. 419–430.

[85] Y. Zhang, "An adaptive flow counting method for anomaly detection in sdn," in *Proceedings of the ninth ACM conference on Emerging networking experiments and technologies*. ACM, 2013, pp. 25–30.

[86] A. Tootoonchian, M. Ghobadi, and Y. Ganjali, "OpenTM: traffic matrix estimator for openflow networks," in *Passive and Active Measurement*. Springer, 2010, pp. 201–210.

[87] G. Liu, M. Trotter, Y. Ren, and T. Wood, "Netalytics: Cloud-scale application performance monitoring with sdn and nfv," in *Proceedings of the 17th International Middleware Conference*, 2016, pp. 1–14.

[88] B. S. Nataraj, S. Khanna, and V. Srinivasan, "Ternary content addressable memory cell," November 2000, uS Patent 6,154,384.

[89] M. Moshref, M. Yu, and R. Govindan, "Resource/accuracy tradeoffs in software-defined measurement," in *Proceedings of ACM HotSDN Workshop*, 2013, pp. 73–78.

[90] W. Stallings, *SNMP, SNMPv2, and CMIP: The practical guide to network management*. Addison-Wesley Longman Publishing Co., 1993.

[91] A. Gupta, R. Harrison, M. Canini, N. Feamster, J. Rexford, and W. Willinger, "Sonata: Query-driven streaming network telemetry," in *ACM SIGCOMM*, 2018, p. 357–371.

[92] F. Paolucci, A. Sgambelluri, F. Cugini, and P. Castoldi, "Network telemetry streaming services in sdn-based disaggregated optical networks," *Journal of Lightwave Technology*, vol. 36, no. 15, pp. 3142–3149, 2018.

[93] H. Song, F. Qin, P. Martinez-Julia, L. Ciavaglia, and A. Wang, "Network telemetry framework," Working Draft, IETF Secretariat, Internet-Draft draft-ietf-opsawg-ntf-04, September 2020. [Online]. Available: http://www.ietf.org/internet-drafts/draft-ietf-opsawg-ntf-04.txt

[94] T. P. A. W. Group, "In-band Network Telemetry (INT) data plane specification," June 2020. [Online]. Available: https://github.com/p4lang/p4-applications/blob/master/docs/INT_v2_1.pdf

[95] "Barefoot deep insight™– solution brief," Barefoot Networks, White paper, 2018. [Online]. Available: https://www.barefootnetworks.com/static/app/pdf/DI-UG42-003ea-ProdBrief.pdf

[96] Y. Li, R. Miao, H. H. Liu, Y. Zhuang, F. Feng, L. Tang, Z. Cao, M. Zhang, F. Kelly, M. Alizadeh, and M. Yu, "HPCC: High precision congestion control," in *ACM SIGCOMM*, 2019, p. 44–58.

[97] "Cisco streaming telemetry." [Online]. Available: https://developer.cisco.com/docs/ios-xe/#!streaming-telemetry-quick-start-guide/streaming-telemetry

[98] "Int using netronome agilio cx smartnic." [Online]. Available: https://www.netronome.com/blog/in-band-network-telemetry-its-not-rocket-science/

[99] Y. Feng, S. Panda, S. G. Kulkarni, K. K. Ramakrishnan, and N. Duffield, "A smartnic-accelerated monitoring platform for in-band network telemetry," in *IEEE LANMAN*, 2020, pp. 1–6.

[100] "Making the switch: Disruptive telecom white box collaboration accelerates and opens the platform, powering unprecedented network performance and insights," April 2017. [Online]. Available: https://about.att.com/story/white_box_collaboration.html

[101] S. Tang, D. Li, B. Niu, J. Peng, and Z. Zhu, "Sel-int: A runtime-programmable selective in-band network telemetry system," *IEEE Transactions on Network and Service Management*, vol. 17, no. 2, pp. 708–721, 2020.

[102] R. Ben Basat, S. Ramanathan, Y. Li, G. Antichi, M. Yu, and M. Mitzenmacher, "Pint: Probabilistic in-band network telemetry," in *Proceedings of ACM SIGCOMM Conference*, 2020, p. 662–680.

[103] N. Dragoni, S. Giallorenzo, A. L. Lafuente, M. Mazzara, F. Montesi, R. Mustafin, and L. Safina, "Microservices: yesterday, today, and tomorrow," in *Present and Ulterior Software Engineering*. Springer, 2017, pp. 195–216.

[104] "ITU-t recommendation g.709/y.1331: Interfaces for the optical transport network," Technical Report, 2016. [Online]. Available: http://www.itu.int/rec/T-REC-G.709/

[105] P. Gill, N. Jain, and N. Nagappan, "Understanding network failures in data centers: measurement, analysis, and implications," in *ACM SIGCOMM CCR*, vol. 41, no. 4. ACM, 2011, pp. 350–361.

[106] A. Markopoulou, G. Iannaccone, S. Bhattacharyya, C.-N. Chuah, Y. Ganjali, and C. Diot, "Characterization of failures in an operational ip backbone network," *IEEE/ACM Transactions on Networking (TON)*, vol. 16, no. 4, pp. 749–762, 2008.

[107] R. Krauthgamer, J. S. Naor, and R. Schwartz, "Partitioning graphs into balanced components," in *Proc. of SODA*, 2009, pp. 942–949.

[108] Y. Dinitz, N. Garg, and M. X. Goemans, "On the single-source unsplittable flow problem," *Combinatorica*, vol. 19, no. 1, pp. 17–41, 1999.

[109] B. Lantz, B. Heller, and N. McKeown, "A network in a laptop: rapid prototyping for software-defined networks," in *ACM HotNets*, 2010, pp. 1–6.

[110] *P4 Language Consortium. 2018. Behavioral Model (BMv2)*, 2018. [Online]. Available: https://github.com/p4lang/behavioral-model

[111] "NIST Definition of Microservices, Application Containers and System Virtual Machines," NIST Special Publication 800-180 (DRAFT), National Institute of Standards

and Technology (NIST), Feb 2016. [Online]. Available: http://csrc.nist.gov/publications/drafts/800-180/sp800-180_draft.pdf

[112] N. Dragoni, S. Giallorenzo, A. L. Lafuente, M. Mazzara, F. Montesi, R. Mustafin, and L. Safina, "Microservices: yesterday, today, and tomorrow," in *Present and Ulterior Software Engineering*. Springer, 2017, pp. 195–216.

[113] Surendra, M. Tufail, S. Majee, C. Captari, and S. Homma, "Service function chaining use cases in data centers," Working Draft, IETF Secretariat, Internet-Draft draft-ietf-sfc-dc-use-cases-06, February 2017. [Online]. Available: https://tools.ietf.org/html/draft-ietf-sfc-dc-use-cases-06

[114] "Blue coat® systems proxysg™," Tech. Report, Blue Coat® Systems, Tech. Report. [Online]. Available: https://bto.bluecoat.com/sites/default/files/tech_pubs/SGOS_4.3.1_Upgrade_Downgrade.pdf

[115] "Barracuda web application firewall." [Online]. Available: https://www.barracuda.com/products/webapplicationfirewall

[116] R. Morris, E. Kohler, J. Jannotti, and M. F. Kaashoek, "The click modular router," *ACM SIGOPS Operating Systems Review*, vol. 33, no. 5, pp. 217–231, December 1999.

[117] "Network Functions Virtualisation (NFV); Virtual Network Functions Architecture," White paper, Dec 2014, accessed: Feb 05, 2017. [Online]. Available: http://www.etsi.org/deliver/etsi_gs/NFV-SWA/001_099/001/01.01.01_60/gs_NFV-SWA001v010101p.pdf

[118] C. Dumitrescu, "Design patterns for packet processing applications on multi-core intel architecture processors." White Paper, December 2008.

[119] J. Martins, M. Ahmed, C. Raiciu, V. Olteanu, M. Honda, R. Bifulco, and F. Huici, "ClickOS and the art of network function virtualization," in *Proceedings of USENIX NSDI*, 2014, pp. 459–473.

[120] "Receiver side scaling," https://www.kernel.org/doc/Documentation/networking/scaling.txt. [Online]. Available: https://www.kernel.org/doc/Documentation/networking/scaling.txt

[121] "TOSCA Simple Profile for Network Functions Virtualization (NFV) Version 1.0," Committee Specification Draft 03, Mar 2014, accessed: Apr 09, 2017. [Online]. Available: https://docs.oasis-open.org/tosca/tosca-nfv/v1.0/csd03/tosca-nfv-v1.0-csd03.pdf

[122] R. Penno, P. Quinn, D. Zhou, and J. Li, "Yang data model for service function chaining," Working Draft, IETF Secretariat, Internet-Draft draft-penno-sfc-yang-15, June 2016, http://www.ietf.org/internet-drafts/draft-penno-sfc-yang-15.txt. [Online]. Available: http://www.ietf.org/internet-drafts/draft-penno-sfc-yang-15.txt

[123] S. Han, K. Jang, A. Panda, S. Palkar, D. Han, and S. Ratnasamy, "SoftNIC: A software NIC to augment hardware," *Dept. EECS, Univ. California, Berkeley, Berkeley, CA, USA, Tech. Rep. UCB/EECS-2015-155*, 2015.

[124] "Network Functions Virtualisation (NFV); Management and Orchestration ," White paper, Dec 2014, accessed: Apr 09, 2017. [Online]. Available: http://www.etsi.org/deliver/etsi_gs/NFV-MAN/001_099/001/01.01.01_60/gs_NFV-MAN001v010101p.pdf

[125] Y. Zhang, B. Anwer, V. Gopalakrishnan, B. Han, J. Reich, A. Shaikh, and Z.-L. Zhang, "Parabox: Exploiting parallelism for virtual network functions in service chaining," in *Proceedings of ACM SOSR*, 2017, pp. 143–149.

[126] C. Sun, J. Bi, Z. Zheng, H. Yu, and H. Hu, "Nfp: Enabling network function parallelism in nfv," in *Proceedings of ACM SIGCOMM*, 2017, pp. 43–56.

[127] G. Liu, Y. Ren, M. Yurchenko, K. Ramakrishnan, and T. Wood, "Microboxes: high performance nfv with customizable, asynchronous tcp stacks and dynamic subscriptions," in *Proceedings of ACM SIGCOMM Conference*, 2018, pp. 504–517.

[128] "Intel data path development kit," http://dpdk.org/. [Online]. Available: http://dpdk.org/

[129] "hugetlbfs kernel documentation." [Online]. Available: https://www.kernel.org/doc/Documentation/vm/hugetlbpage.txt

[130] J. C. Mogul and K. Ramakrishnan, "Eliminating receive livelock in an interrupt-driven kernel," *ACM Transactions on Computer Systems*, vol. 15, no. 3, pp. 217–252, 1997.

[131] C. Dovrolis, B. Thayer, and P. Ramanathan, "Hip: hybrid interrupt-polling for the network interface," *ACM SIGOPS Operating Systems Review*, vol. 35, no. 4, pp. 50–60, 2001.

[132] J. Hwang, K. Ramakrishnan, and T. Wood, "NetVM: high performance and flexible networking using virtualization on commodity platforms," in *Proceedings of USENIX NSDI*, 2014, pp. 445–458.

[133] "Kernel address space layout randomization." [Online]. Available: https://www.kernel.org/doc/html/v4.13/security/self-protection.html

[134] S. G. Kulkarni, W. Zhang, J. Hwang, S. Rajagopalan, K. Ramakrishnan, T. Wood, M. Arumaithurai, and X. Fu, "Nfvnice: Dynamic backpressure and scheduling for nfv service chains," in *Proceedings of ACM SIGCOMM*, 2017, pp. 71–84.

[135] "Cfs scheduler," https://www.kernel.org/doc/Documentation/scheduler/sched-design-CFS.txt. [Online]. Available: https://www.kernel.org/doc/Documentation/scheduler/sched-design-CFS.txt

[136] "Real-time group scheduling," https://www.kernel.org/doc/Documentation/scheduler/sched-rt-group.txt. [Online]. Available: https://www.kernel.org/doc/Documentation/scheduler/sched-rt-group.txt

[137] "pktgen-dpdk," http://git.dpdk.org/apps/pktgen-dpdk/. [Online]. Available: http://git.dpdk.org/apps/pktgen-dpdk/

[138] P. Emmerich, S. Gallenmüller, D. Raumer, F. Wohlfart, and G. Carle, "Moongen: A scriptable high-speed packet generator," in *Proceedings of ACM IMC*. ACM, 2015, pp. 275–287.

[139] C. Sieber, R. Durner, M. Ehm, W. Kellerer, and P. Sharma, "Towards optimal adaptation of nfv packet processing to modern cpu memory architectures," in *Proceedings of the 2nd Workshop on Cloud-Assisted Networking*. ACM, 2017, pp. 7–12.

[140] "Netbricks repository," https://github.com/NetSys/NetBricks. [Online]. Available: https://github.com/NetSys/NetBricks

[141] A. Roy, H. Zeng, J. Bagga, G. Porter, and A. C. Snoeren, "Inside the social network's (datacenter) network," in *ACM SIGCOMM Computer Communication Review*, vol. 45, no. 4. ACM, 2015, pp. 123–137.

[142] T. Barbette, C. Soldani, and L. Mathy, "Fast userspace packet processing," in *Proceedings of ACM/IEEE ANCS*, 2015, pp. 5–16.

[143] W. Sun and R. Ricci, "Fast and flexible: Parallel packet processing with gpus and click," in *Proceedings of ACM/IEEE ANCS*, 2013, pp. 25–36.

[144] J. Kim, K. Jang, K. Lee, S. Ma, J. Shim, and S. Moon, "NBA (network balancing act): a high-performance packet processing framework for heterogeneous processors," in *Proceedings of ACM EuroSys*, 2015, pp. 22:1–22:14.

[145] B. Li, K. Tan, L. L. Luo, Y. Peng, R. Luo, N. Xu, Y. Xiong, P. Cheng, and E. Chen, "Clicknp: Highly flexible and high performance network processing with reconfigurable hardware," in *Proceedings of ACM SIGCOMM Conference*, 2016, pp. 1–14.

[146] M. Gallo and R. Laufer, "Clicknf: a modular stack for custom network functions," in *Proceedings of USENIX ATC*, 2018, pp. 745–757.

[147] M. A. Jamshed, Y. Moon, D. Kim, D. Han, and K. Park, "mOS: A reusable networking stack for flow monitoring middleboxes," in *Proceedings of USENIX NSDI*, 2017, pp. 113–129.

[148] B. Anwer, T. Benson, N. Feamster, and D. Levin, "Programming slick network functions," in *Proceedings of ACM SOSR*, 2015, pp. 14:1–14:13.

[149] R. Kawashima and H. Matsuo, "A generic and efficient local service function chaining framework for user VM-dedicated micro-VNFs," *IEICE Transactions on Communications*, vol. E100.B, pp. 2017–2026, 2017.

[150] A. Madhavapeddy, R. Mortier, C. Rotsos, D. Scott, B. Singh, T. Gazagnaire, S. Smith, S. Hand, and J. Crowcroft, "Unikernels: Library operating systems for the cloud," *ACM SIGPLAN Notices*, vol. 48, no. 4, pp. 461–472, April 2013.

[151] P. L. Ventre, P. Lungaroni, G. Siracusano, C. Pisa, F. Schmidt, F. Lombardo, and S. Salsano, "On the fly orchestration of unikernels: Tuning and performance evaluation of virtual infrastructure managers," *IEEE Transactions on Cloud Computing*, 2018 (Early Access).

[152] "Clearwater IMS," Project documentation, October 2018. [Online]. Available: https://media.readthedocs.org/pdf/clearwater/latest/clearwater.pdf

[153] R. Mijumbi, S. Hasija, S. Davy, A. Davy, B. Jennings, and R. Boutaba, "Topology-aware prediction of virtual network function resource requirements," *IEEE Transactions on Network and Service Management*, vol. 14, no. 1, pp. 106–120, March 2017.

[154] J. Duan, C. Wu, F. Le, A. X. Liu, and Y. Peng, "Dynamic scaling of virtualized, distributed service chains: A case study of ims," *IEEE Journal on Selected Areas in Communications*, vol. 35, no. 11, pp. 2501–2511, November 2017.

[155] M. T. Raza, S. Lu, M. Gerla, and X. Li, "Refactoring network functions modules to reduce latencies and improve fault tolerance in nfv," *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 10, pp. 2275–2287, Oct 2018.

[156] F. Rambach, B. Konrad, L. Dembeck, U. Gebhard, M. Gunkel, M. Quagliotti, L. Serra, and V. Lopez, "A multilayer cost model for metro/core networks," *IEEE/OSA Journal of Optical Communications and Networking*, vol. 5, no. 3, pp. 210–225, March 2013.

[157] E. Modiano, "Traffic grooming in wdm networks," *IEEE Communications Magazine*, vol. 39, no. 7, pp. 124–129, July 2001.

[158] K. Zhu and B. Mukherjee, "A review of traffic grooming in wdm optical networks: Architectures and challenges," *Optical Networks Magazine*, vol. 4, no. 2, pp. 55–64, 2003.

[159] D. BianchiAn, G. Parthasarathy, and Y. Xu, "Otn system and method for supporting single-fiber bidirectional transmission of supervisory channel light," September 2015, wO Application WO2015127780. [Online]. Available: https://patents.google.com/patent/WO2015127780A1/en

[160] S. Chen, J. Liu, Y. Zhao, L. Zhu, A. Wang, S. Li, J. Du, C. Du, Q. Mo, and J. Wang, "Full-duplex bidirectional data transmission link using twisted lights multiplexing over 1.1-km orbital angular momentum fiber," *Scientific reports*, vol. 6, p. 38181, 2016.

[161] "Link aggregation control protocol." [Online]. Available: http://www.ieee802.org/3/ad/public/mar99/seaman_1_0399.pdf

[162] Y. Dinitz, N. Garg, and M. X. Goemans, "On the single-source unsplittable flow problem," in *Proceedings of IEEE FOCS*, 1998, pp. 290–299.

[163] J. Edmonds and R. M. Karp, "Theoretical improvements in algorithmic efficiency for network flow problems," *Journal of the ACM*, vol. 19, no. 2, pp. 248–264, April 1972.

[164] N. Spring, R. Mahajan, and D. Wetherall, "Measuring ISP topologies with Rocketfuel," in *Proceedings of ACM SIGCOMM Conference*, 2002, pp. 133–145.

[165] P. Erdös and A. Rényi, "On random graphs, i," *Publicationes Mathematicae (Debrecen)*, vol. 6, pp. 290–297, 1959.

[166] M. R. Rahman and R. Boutaba, "SVNE: Survivable virtual network embedding algorithms for network virtualization," *IEEE Transactions on Network and Service Management*, vol. 10, no. 2, pp. 105–118, June 2013.

[167] S. R. Chowdhury, R. Ahmed, M. M. Alam Khan, N. Shahriar, R. Boutaba, J. Mitra, and F. Zeng, "Dedicated protection for survivable virtual network embedding," *IEEE Transactions on Network and Service Management*, vol. 13, no. 4, pp. 913–926, December 2016.

[168] M. Savi, C. Rozic, C. Matrakidis, D. Klonidis, D. Siracusa, and I. Tomkos, "Benefits of multi-layer application-aware resource allocation and optimization," in *Proc. of IEEE EuCNC*, 2017, pp. 1–5.

[169] Y. Li, H. Li, Y. Liu, and Y. Ji, "Multi-layer service function chaining scheduling based on auxiliary graph in ip over optical network," in *Proc. of SPIE*, 2017, pp. 10 464 – 10 464 – 10.

[170] R. Nejabati, E. Escalona, S. Peng, and D. Simeonidou, "Optical network virtualization," in *Proceedings of Optical Network Design and Modeling*, Feb 2011, pp. 1–5.

[171] Ć. Rožić, D. Klonidis, and I. Tomkos, "A survey of multi-layer network optimization," in *Proceedings of Optical Network Design and Modeling*, 2016, pp. 1–6.

[172] M. Duelli, E. Weber, and M. Menth, "A generic algorithm for capex-aware multi-layer network design," in *Proceedings of ITG Symposium on Photonic Networks*, 2009, pp. 1–8.

[173] I. Katib and D. Medhi, "IP/MPLS-over-OTN-over-DWDM multilayer networks: an integrated three-layer capacity optimization model, a heuristic, and a study," *IEEE Transactions on Network and Service Management*, vol. 9, no. 3, pp. 240–253, September 2012.

[174] C. Govardan, S. R. DS, C. Jagadeesh, R. Gowrishankar, P. Beherey, B. S. Kishorez *et al.*, "A heuristic algorithm for network optimization of otn over dwdm network," in *Proc. of IEEE ANTS*, 2015, pp. 1–6.

[175] E. Palkopoulou, D. A. Schupke, and T. Bauschert, "Energy efficiency and capex minimization for backbone network planning: Is there a tradeoff?" in *Proc. of IEEE ANTS*, 2009, pp. 1–3.

[176] H. Zhang and A. Durresi, "Differentiated multi-layer survivability in IP/WDM networks," in *Proceedings of IEEE/IFIP NOMS*, 2002, pp. 681–694.

[177] W. Bigos, B. Cousin, S. Gosselin, M. Le Foll, and H. Nakajima, "Survivable MPLS over optical transport networks: Cost and resource usage analysis," *IEEE Journal on Selected Areas in Communications*, vol. 25, no. 5, pp. 949–962, June 2007.

[178] W. Lu, X. Yin, X. Cheng, and Z. Zhu, "On cost-efficient integrated multilayer protection planning in ip-over-eons," *Journal of Lightwave Technology*, vol. 36, no. 10, pp. 2037–2048, 2018.

[179] A. Alashaikh, D. Tipper, and T. Gomes, "Supporting differentiated resilience classes in multilayer networks," in *Proceedings of IEEE DRCN*, 2016, pp. 31–38.

[180] M. Tornatore, D. Lucerna, B. Mukherjee, and A. Pattavina, "Multilayer protection with availability guarantees in optical wdm networks," *Journal of Network and Systems Management*, vol. 20, no. 1, pp. 34–55, March 2012.

[181] O. Gerstel, C. Filsfils, T. Telkamp, M. Gunkel, M. Horneffer, V. Lopez, and A. Mayoral, "Multi-layer capacity planning for ip-optical networks," *IEEE Communications Magazine*, vol. 52, no. 1, pp. 44–51, January 2014.

[182] P. Demeester, M. Gryseels, A. Autenrieth, C. Brianza, L. Castagna, G. Signorelli, R. Clemenfe, M. Ravera, A. Jajszczyk, D. Janukowicz, K. Van Doorselaere, and Y. Harada, "Resilience in multilayer networks," *IEEE Communications Magazine*, vol. 37, no. 8, pp. 70–76, August 1999.

[183] M. Melo, J. Carapinha, S. Sargento, L. Torres, P. N. Tran, U. Killat, and A. Timm-Giel, "Virtual network mapping–an optimization problem," in *Mobile Networks and Management*. Springer, 2012, pp. 187–200.

[184] A. Anagnostopoulos, F. Grandoni, S. Leonardi, and A. Wiese, "A mazing 2+ ε approximation for unsplittable flow on a path," in *Proceedings of ACM-SIAM SODA 2014*. SIAM, 2014, pp. 26–41.

[185] P. Bonsma, J. Schulz, and A. Wiese, "A constant factor approximation algorithm for unsplittable flow on paths," in *Proceedings of IEEE FOCS 2011*, 2011, pp. 47–56.

[186] A. Buluç, H. Meyerhenke, I. Safro, P. Sanders, and C. Schulz, "Recent advances in graph partitioning," in *Algorithm Engineering: Selected Results and Surveys, LNCS 9220*. Springer-Verlag, 2015 (in press).

[187] Y. Zhu and M. H. Ammar, "Algorithms for assigning substrate network resources to virtual network components." in *INFOCOM*, vol. 1200, no. 2006, 2006, pp. 1–12.

[188] M. R. Rahman and R. Boutaba, "SVNE: Survivable Virtual Network Embedding Algorithms for Network Virtualization," *Network and Service Management, IEEE Transactions on*, vol. 10, no. 2, pp. 105–118, 2013.

[189] J. W. Suurballe, "Disjoint paths in a network," *Networks*, vol. 4, no. 2, pp. 125–145, 1974.

[190] M. Kurant and P. Thiran, "Survivable mapping algorithm by ring trimming (smart) for large ip-over-wdm networks," in *IEEE BroadNets*, 2004, pp. 44–53.

[191] Z. Zhou, T. Lin, K. Thulasiraman, G. Xue, and S. Sahni, "Novel survivable logical topology routing in ip-over-wdm networks by logical protecting spanning tree set," in *IEEE ICUMT*, 2012, pp. 650–656.

[192] C. Liu and L. Ruan, "A new survivable mapping problem in ip-over-wdm networks," *Selected Areas in Communications, IEEE Journal on*, vol. 25, no. 3, pp. 25–34, 2007.

[193] M. Kurant and P. Thiran, "Survivable routing of mesh topologies in ip-over-wdm networks by recursive graph contraction," *Selected Areas in Communications, IEEE Journal on*, vol. 25, no. 5, pp. 922–933, 2007.

[194] H. Yu, C. Qiao, V. Anand, X. Liu, H. Di, and G. Sun, "Survivable virtual infrastructure mapping in a federated computing and networking system under single regional failures," in *Proc. of IEEE GLOBECOM*, 2010, pp. 1–6.

[195] H. Yu, V. Anand, C. Qiao, and G. Sun, "Cost efficient design of survivable virtual infrastructure to recover from facility node failures," in *Proc. of IEEE ICC*, 2011, pp. 1–6.

[196] X. Liu, Y. Wang, A. Xiao, X. Qiu, and W. Li, "Disaster-prediction based virtual network mapping against multiple regional failures," in *Proceedings of IEEE/IFIP IM 2015*. IEEE, 2015, pp. 371–378.

[197] M. Pourvali, K. Liang, F. Gu, H. Bai, K. Shaban, S. Khan, and N. Ghani, "Progressive recovery for network virtualization after large-scale disasters," in *ICNC 2016*, 2016.

[198] B. Guo, C. Qiao, J. Wang, H. Yu, Y. Zuo, J. Li, Z. Chen, and Y. He, "Survivable virtual network design and embedding to survive a facility node failure," *Lightwave Technology, Journal of*, vol. 32, no. 3, pp. 483–493, 2014.

[199] T. Guo, N. Wang, K. Moessner, and R. Tafazolli, "Shared backup network provision for virtual network embedding," in *IEEE International Conference on Communications (ICC)*. IEEE, 2011, pp. 1–5.

[200] M. M. A. Khan, N. Shahriar, R. Ahmed, and R. Boutaba, "SiMPLE: Survivability in multipath link embedding," in *ACM/IEEE/IFIP CNSM 2015, Barcelona, Spain, November 9-13, 2015*, 2015, pp. 210–218.

[201] M. G. Rabbani, M. F. Zhani, and R. Boutaba, "On achieving high survivability in virtualized data centers," *IEICE Transactions on Communications*, vol. 97, no. 1, pp. 10–18, 2014.

[202] Q. Zhang, M. F. Zhani, M. Jabri, and R. Boutaba, "Venice: Reliable virtual data center embedding in clouds," in *Proc. of IEEE INFOCOM 2014, Toronto, Canada*. IEEE, 2014, pp. 289–297.

[203] H. Jiang, Y. Wang, L. Gong, and Z. Zhu, "Availability-aware survivable virtual network embedding in optical datacenter networks," *Optical Communications and Networking, IEEE/OSA Journal of*, vol. 7, no. 12, pp. 1160–1171, 2015.

[204] H. Jiang, L. Gong, and Z. Zuqing, "Efficient joint approaches for location-constrained survivable virtual network embedding," in *IEEE GLOBECOM*, 2014, pp. 1810–1815.

[205] N. Shahriar, S. Taeb, S. R. Chowdhury, M. Zulfiqar, M. Tornatore, R. Boutaba, J. Mitra, and M. Hemmati, "Reliable slicing of 5g transport networks with bandwidth squeezing and multi-path provisioning," *IEEE Transactions on Network and Service Management*, vol. 17, no. 3, pp. 1418–1431, 2020.

[206] M. Skutella, "Approximating the Single Source Unsplittable Min-cost Flow Problem," *Mathematical Programming*, vol. 91, no. 3, pp. 493–514, 2002.

[207] Z. Friggstad and Z. Gao, "On linear programming relaxations for unsplittable flow in trees," in *LIPIcs-Leibniz International Proceedings in Informatics*, vol. 40. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2015.

[208] A. Yassine, H. Rahimi, and S. Shirmohammadi, "Software defined network traffic measurement: Current trends and challenges," *IEEE Instrumentation & Measurement Magazine*, vol. 18, no. 2, pp. 42–50, April 2015.

[209] C. Yu, C. Lumezanu, Y. Zhang, V. Singh, G. Jiang, and H. V. Madhyastha, "FlowSense: Monitoring Network Utilization with Zero Measurement Cost," in *Passive and Active Measurement*. Springer, 2013, pp. 31–41.

[210] P. Bosshart, G. Gibb, H.-S. Kim, G. Varghese, N. McKeown, M. Izzard, F. Mujica, and M. Horowitz, "Forwarding metamorphosis: Fast programmable match-action processing in hardware for sdn," in *Proceedings of ACM SIGCOMM Conference*, 2013, pp. 99–110.

[211] "Floodlight openflow controller," http://www.projectfloodlight.org/floodlight/.

[212] "Payless github repository." [Online]. Available: https://github.com/srcvirus/floodlight/tree/master/src/main/java/net/floodlightcontroller/netmonitor

[213] A. Roy, H. Zeng, J. Bagga, G. Porter, and A. C. Snoeren, "Inside the social network's (datacenter) network," in *Proceedings of ACM SIGCOMM Conference*, 2015, p. 123–137.

[214] T. Benson, A. Akella, and D. A. Maltz, "Network traffic characteristics of data centers in the wild," in *Proceedings of the ACM Internet Measurement Conference*, 2010, p. 267–280.

[215] "The CAIDA UCSD anonymized internet traces - 2016 - 2016/04/06 13:19:00 utc." [Online]. Available: https://www.caida.org/data/passive/passive_dataset.xml

[216] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," in *ACM SIGCOMM*, 2008, p. 63–74.

[217] K. Bakhshaliyev, M. A. Canbaz, and M. H. Gunes, "Investigating characteristics of internet paths," *ACM Trans. Model. Perform. Eval. Comput. Syst.*, vol. 4, no. 3, September 2019.

[218] "Introducing data center fabric, the next-generation facebook data center network," November 2014. [Online]. Available: https://engineering.fb.com/production-engineering/introducing-data-center-fabric-the-next-generation-facebook-data-center-network/

[219] A. Deshpande, C. Guestrin, S. R. Madden, J. M. Hellerstein, and W. Hong, "Model-driven data acquisition in sensor networks," in *VLDB*, 2004, pp. 588–599.

[220] D. Goldsmith and J. Brusey, "The spanish inquisition protocol—model based transmission reduction for wireless sensor networks," in *IEEE SENSORS*, 2010, pp. 2043–2048.

[221] G. M. Dias, B. Bellalta, and S. Oechsner, "A survey about prediction-based data reduction in wireless sensor networks," *ACM Comput. Surv.*, vol. 49, no. 3, Nov. 2016.

[222] N. K. Sharma, A. Kaufmann, T. Anderson, A. Krishnamurthy, J. Nelson, and S. Peter, "Evaluating the power of flexible packet processing for network resource allocation," in *USENIX NSDI*, 2017, pp. 67–82.

[223] A. C. Lapolli, J. Adilson Marques, and L. P. Gaspary, "Offloading real-time ddos attack detection to programmable data planes," in *IFIP/IEEE IM*, 2019, pp. 19–27.

[224] X. Jin, X. Li, H. Zhang, R. Soulé, J. Lee, N. Foster, C. Kim, and I. Stoica, "Netcache: Balancing key-value stores with fast in-network caching," in *ACM SOSP*, 2017, p. 121–136.

[225] V. Sivaraman, S. Narayana, O. Rottenstreich, S. Muthukrishnan, and J. Rexford, "Heavy-hitter detection entirely in the data plane," in *ACM SOSR*, 2017, pp. 164–176.

[226] J. Sonchack, A. J. Aviv, E. Keller, and J. M. Smith, "Turboflow: Information rich flow record generation on commodity switches," in *Proceedings of ACM EuroSys*. ACM, 2018, pp. 11:1–11:16.

[227] Z. Xiong and N. Zilberman, "Do switches dream of machine learning? toward in-network classification," in *ACM HotNets*, 2019, p. 25–33.

[228] J. S. Hunter, "The exponentially weighted moving average," *Journal of quality technology*, vol. 18, no. 4, pp. 203–210, 1986.

[229] "Traffic Monitoring using sFlow," http://www.sflow.org/.

[230] 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, January 2015.

[231] P. Tsai, C. Tsai, C. Hsu, and C. Yang, "Network monitoring in software-defined networking: A review," *IEEE Systems Journal*, vol. 12, no. 4, pp. 3958–3969, December 2018.

[232] Y. Gong, X. Wang, M. Malboubi, S. Wang, S. Xu, and C.-N. Chuah, "Towards accurate online traffic matrix estimation in software-defined networks," in *Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research*, 2015.

[233] Y. Tian, W. Chen, and C. Lea, "An sdn-based traffic matrix estimation framework," *IEEE Transactions on Network and Service Management*, vol. 15, no. 4, pp. 1435–1445, 2018.

[234] L. Jose, M. Yu, and J. Rexford, "Online measurement of large traffic aggregates on commodity switches," in *Proc. of the USENIX HotICE workshop*, 2011.

[235] L. Yang, B. Ng, and W. K. G. Seah, "Heavy hitter detection and identification in software defined networking," in *2016 25th International Conference on Computer Communication and Networks (ICCCN)*, 2016, pp. 1–10.

[236] Y. Afek, A. Bremler-Barr, S. Landau Feibish, and L. Schiff, "Detecting heavy flows in the sdn match and action model," *Computer Networks*, vol. 136, pp. 1 – 12, 2018.

[237] K. Phemius and M. Bouet, "Monitoring latency with openflow," in *Proceedings of the 9th International Conference on Network and Service Management (CNSM 2013)*, 2013, pp. 122–125.

[238] A. Atary and A. Bremler-Barr, "Efficient round-trip time monitoring in openflow networks," in *IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications*, 2016, pp. 1–9.

[239] C. Yu, C. Lumezanu, A. Sharma, Q. Xu, G. Jiang, and H. Madhyastha, "Software-defined latency monitoring in data center networks," in *PAM*, 2015, pp. 360–372.

[240] X. Zhang, Y. Wang, J. Zhang, L. Wang, and Y. Zhao, "Ringlm: A link-level packet loss monitoring solution for software-defined networks," *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 8, pp. 1703–1720, 2019.

[241] S. Shin and G. Gu, "Cloudwatcher: Network security monitoring using openflow in dynamic cloud networks (or: How to provide security monitoring as a service in clouds?),"

in *2012 20th IEEE international conference on network protocols (ICNP)*. IEEE, 2012, pp. 1–6.

[242] S. Shirali-Shahreza and Y. Ganjali, "Efficient implementation of security applications in openflow controller with flexam," in *2013 IEEE 21st Annual Symposium on High-Performance Interconnects*, 2013, pp. 49–54.

[243] S. Lee, J. Kim, S. Shin, P. Porras, and V. Yegneswaran, "Athena: A framework for scalable anomaly detection in software-defined networks," in *2017 47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2017, pp. 249–260.

[244] J. Zheng, Q. Li, G. Gu, J. Cao, D. K. Y. Yau, and J. Wu, "Realtime ddos defense using cots sdn switches via adaptive correlation analysis," *IEEE Transactions on Information Forensics and Security*, vol. 13, no. 7, pp. 1838–1853, 2018.

[245] M. Yu, L. Jose, and R. Miao, "Software defined traffic measurement with opensketch," in *Proceedings of USENIX NSDI*, 2013, pp. 29–42.

[246] G. Cormode and S. Muthukrishnan, "An improved data stream summary: the count-min sketch and its applications," *Journal of Algorithms*, vol. 55, no. 1, pp. 58–75, 2005.

[247] L. Tang, Q. Huang, and P. P. Lee, "Mv-sketch: A fast and compact invertible sketch for heavy flow detection in network data streams," in *IEEE INFOCOM*, 2019.

[248] Q. Huang and P. P. C. Lee, "Ld-sketch: A distributed sketching design for accurate and scalable anomaly detection in network data streams," in *IEEE INFOCOM 2014 - IEEE Conference on Computer Communications*, April 2014, pp. 1420–1428.

[249] Q. Huang, X. Jin, P. P. Lee, R. Li, L. Tang, Y.-C. Chen, and G. Zhang, "SketchVisor: Robust network measurement for software packet processing," in *Proceedings of ACM SIGCOMM Conference*, 2017, pp. 113–126.

[250] T. Yang, J. Jiang, P. Liu, Q. Huang, J. Gong, Y. Zhou, R. Miao, X. Li, and S. Uhlig, "Elastic sketch: Adaptive and fast network-wide measurements," in *Proceedings of ACM SIGCOMM Conference*, 2018, pp. 561–575.

[251] Z. Liu, A. Manousis, G. Vorsanger, V. Sekar, and V. Braverman, "One sketch to rule them all: Rethinking network flow monitoring with UnivMon," in *Proceedings of ACM SIGCOMM Conference*, 2016, pp. 101–114.

[252] T. Pan, E. Song, C. Jia, W. Cao, T. Huang, and B. Liu, "Lightweight network-wide telemetry without explicitly using probe packets," in *IEEE INFOCOM Workshops*, 2020, pp. 1354–1355.

[253] C. Jia, T. Pan, Z. Bian, X. Lin, E. Song, C. Xu, T. Huang, and Y. Liu, "Rapid detection and localization of gray failures in data centers via in-band network telemetry," in *Proceedings of IEEE/IFIP NOMS*, 2020, pp. 1–9.

[254] S. Wang, Y. Chen, J. Li, H. Hu, J. Tsai, and Y. Lin, "A bandwidth-efficient int system for tracking the rules matched by the packets of a flow," in *IEEE GLOBECOM*, 2019, pp. 1–6.

[255] P. Janakaraj, P. Pinyoanuntapong, P. Wang, and M. Lee, "Towards in-band telemetry for self driving wireless networks," in *IEEE INFOCOM Workshops 2020*, 2020, pp. 766–773.

[256] M. Anand, R. Subrahmaniam, and R. Valiveti, "Point: An intent-driven framework for integrated packet-optical in-band network telemetry," in *IEEE ICC*, 2018, pp. 1–6.

[257] S. Tang, J. Kong, B. Niu, and Z. Zhu, "Programmable multilayer int: An enabler for ai-assisted network automation," *IEEE Communications Magazine*, vol. 58, no. 1, pp. 26–32, 2020.

[258] B. Niu, J. Kong, S. Tang, Y. Li, and Z. Zhu, "Visualize your ip-over-optical network in realtime: A p4-based flexible multilayer in-band network telemetry (ml-int) system," *IEEE Access*, vol. 7, pp. 82 413–82 423, 2019.

[259] F. Brockners, S. Bhandari, and T. Mizrahi, "Data fields for in-situ oam," Working Draft, IETF Secretariat, Internet-Draft draft-ietf-ippm-ioam-data-10, July 2020. [Online]. Available: http://www.ietf.org/internet-drafts/draft-ietf-ippm-ioam-data-10.txt

[260] A. Jain and E. Y. Chang, "Adaptive sampling for sensor networks," in *Proceeedings of ACM VLDB workshop on Data management for sensor networks*, 2004, pp. 10–16.

[261] B. Gedik, L. Liu, and P. Yu, "Asap: An adaptive sampling approach to data collection in sensor networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 18, no. 12, pp. 1766–1783, December 2007.

[262] J. Kho, A. Rogers, and N. R. Jennings, "Decentralized control of adaptive sampling in wireless sensor networks," *ACM Transactions on Sensor Networks*, vol. 5, no. 3, p. 19, May 2009.

[263] C. Alippi, G. Anastasi, M. Di Francesco, and M. Roveri, "An adaptive sampling algorithm for effective energy management in wireless sensor networks with energy-hungry sensors," *IEEE Transactions on Instrumentation and Measurement*, vol. 59, no. 2, pp. 335–344, February 2010.

[264] R. Willett, A. Martin, and R. Nowak, "Backcasting: adaptive sampling for sensor networks," in *Proceedings of ACM IPSN*, 2004, pp. 124–133.

[265] E. Hernandez, M. Chidester, and A. George, "Adaptive sampling for network management," *Journal of Network and Systems Management*, vol. 9, no. 4, pp. 409–434, 2001.

[266] G. Androulidakis, V. Chatzigiannakis, and S. Papavassiliou, "Network anomaly detection and classification via opportunistic sampling," *Network, IEEE*, vol. 23, no. 1, pp. 6–12, 2009.

[267] N. L. M. van Adrichem, C. Doerr, and F. A. Kuipers, "Opennetmon: Network monitoring in openflow software-defined networks," in *2014 IEEE Network Operations and Management Symposium (NOMS)*, 2014, pp. 1–8.

[268] J. Suh, T. Kwon, C. Dixon, W. Felter, and J. Carter, "Opensample: A low-latency, sampling-based measurement platform for commodity sdn," in *Proceedings of IEEE ICDCS*, 2014, pp. 228–237.

[269] Z. Su, T. Wang, Y. Xia, and M. Hamdi, "Cemon: A cost-effective flow monitoring system in software defined networks," *Computer Networks*, vol. 92, pp. 101–115, 2015.

[270] H. Tahaei, R. Salleh, S. Khan, R. Izard, K.-K. R. Choo, and N. B. Anuar, "A multi-objective software defined network traffic measurement," *Measurement*, vol. 95, pp. 317–327, 2017.

[271] S. Bera, S. Misra, and A. Jamalipour, "Flowstat: Adaptive flow-rule placement for per-flow statistics in sdn," *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 3, pp. 530–539, March 2019.

[272] H. Xu, Z. Yu, C. Qian, X.-Y. Li, Z. Liu, and L. Huang, "Minimizing flow statistics collection cost using wildcard-based requests in sdns," *IEEE/ACM Transactions on Networking (TON)*, vol. 25, no. 6, pp. 3587–3601, 2017.

[273] G. Zhao, H. Xu, J. Fan, L. Huang, and C. Qiao, "Hifi: Hybrid rule placement for fine-grained flow management in sdns," in *Proceedings of IEEE INFOCOM*, 2020, pp. 2341–2350.

[274] T. Pan, E. Song, Z. Bian, X. Lin, X. Peng, J. Zhang, T. Huang, B. Liu, and Y. Liu, "Int-path: Towards optimal path planning for in-band network-wide telemetry," in *Proceedings of IEEE INFOCOM*, 2019, pp. 487–495.

[275] Y. Lin, Y. Zhou, Z. Liu, K. Liu, Y. Wang, M. Xu, J. Bi, Y. Liu, and J. Wu, "Netview: Towards on-demand network-wide telemetry in the data center," *Elsevier Computer Networks*, vol. 180, p. 107386, 2020.

[276] J. A. Marques, M. C. Luizelli, R. I. T. da Costa Filho, and L. P. Gaspary, "An optimization-based approach for efficient network monitoring using in-band network telemetry," *Springer Journal of Internet Services and Applications*, vol. 10, no. 1, p. 12, 2019.

[277] A. Friedley, T. Hoefler, G. Bronevetsky, A. Lumsdaine, and C.-C. Ma, "Ownership passing: Efficient distributed memory programming on multi-core systems," *ACM SIGPLAN Notices*, vol. 48, no. 8, pp. 177–186, February 2013.

[278] G. Zhang, M. De Leenheer, A. Morea, and B. Mukherjee, "A survey on OFDM-based elastic core optical networking," *IEEE Communications Surveys & Tutorials*, vol. 15, no. 1, pp. 65–87, First quarter 2013.

[279] N. Shahriar, S. Taeb, S. R. Chowdhury, M. Tornatore, R. Boutaba, J. Mitra, and M. Hemmati, "Achieving a fully-flexible virtual network embedding in elastic optical networks," in *Proceedings of IEEE INFOCOM*, 2019, pp. 1756–1764.