APPLICATION-AWARENESS IN SOFTWARIZED NETWORKS: BUILDING INTELLIGENT NETWORKS THROUGH APPLICATION AND NETWORK-LAYER COLLABORATION

by

Deepak Nadig

A DISSERTATION

Presented to the Faculty of

The Graduate College at the University of Nebraska

In Partial Fulfilment of Requirements

For the Degree of Doctor of Philosophy

Major: Engineering

(Computer Engineering–Computer Science)

Under the Supervision of Professor Byrav Ramamurthy

Lincoln, Nebraska

June, 2021

APPLICATION-AWARENESS IN SOFTWARIZED NETWORKS: BUILDING INTELLIGENT NETWORKS THROUGH APPLICATION AND NETWORK-LAYER COLLABORATION

Deepak Nadig, Ph.D.

University of Nebraska, 2021

Adviser: Byrav Ramamurthy

Increasingly, campus networks manage a multitude of large-scale data transfers. Big data plays a pivotal role in university research and impacts engineering, agriculture, natural sciences, and humanities. Campus network infrastructures support multiple network management goals, including commodity internet traffic and high-performance networks for scientific research. These goals often impose conflicting requirements on network design and management, and therefore, networks optimized and specially engineered for data-intensive tasks are necessary. Further, many aspects of campus networks are hard to change without impacting regular network operation. Over the years, numerous solutions have focused on the management and security of large-scale data transfers. These solutions severely degrade data transfer performance or result in data flows completely bypassing the campus network management and security controls.

This dissertation will study application-aware architectures and present software defined networking (SDN) and network functions virtualization (NFV) solutions for data-intensive science. Our proposed application-aware SDN solutions span network monitoring, management, service differentiation, and security for data-intensive applications. We first propose a novel application-aware architecture called SNAG (SDN-managed Network Architecture for GridFTP transfers).

SNAG combines application-awareness with SDN-enabled network management to classify, monitor and manage network resources actively. At HCC, we also demonstrate how our system ensures the quality of service (QoS) for high-throughput workflows such as Compact Muon Solenoid (CMS) and Laser Interferometer Gravitational-Wave Observatory (LIGO). Next, we develop a novel application-aware flow reduction (AAFR) algorithm to optimally map service function chains (SFC) to multiple data centers while adhering to the data center's capacity constraints. We then present an application-aware intelligent load balancing system for high-throughput, distributed computing workflows. Our solution integrates with a major U.S. CMS Tier-2 site. Lastly, by developing a scalable application-aware edge computing framework, we focus on building reliable service-to-service communication across distributed infrastructures using a service mesh architecture. By building application-aware architectures and evolving data-intensive applications to collaboratively and securely share application-layer metadata with the network-layer, we pave the way for intelligent networks that are secure, automated, dynamically composable and highly scalable.

DEDICATION

To my wife, Sneha, thank you for your love, encouragement and dedication.

To Manjula and Anantha, my parents, for teaching me the value of education and hard work.

To my late father-in-law, K.G. Shanthappa, your fierce dedication to your profession was a constant source of inspiration to us all.

ACKNOWLEDGMENTS

I want to express my most profound appreciation and gratitude to my advisor Dr. Byrav Ramamurthy. He has been a great mentor, and I am truly fortunate to have had the opportunity to work with him. His consistent guidance and invaluable support have made this a thoughtful and rewarding journey. I want to express my sincere thanks to my committee members, Dr. Lisong Xu, Dr. Mehmet Can Vuran and Dr. Yi Qian. Their timely feedback and suggestions helped improve the quality of this work. I also want to thank my fellow graduate students—those who have moved on, those in a dilemma, and those just beginning—for their support, feedback, and friendship. Specifically, I want to thank Dr. Sara El Alaoui, Dr. Mohammad Alhowaidi and Sampashree Nayak for their assistance and suggestions during the various stages of this work.

Others in academia, research labs, and industry have helped improve the quality of my research. In particular, I want to thank Dr. Brian Bockelman (Morgridge Institute for Research, UW-Madison) and Garhan Attebury (Holland Computing Center, UNL) for many thought-provoking discussions, feedback and insight. I was also very fortunate to receive invaluable support and encouragement from the late Dr. David Swanson during the early stages of my Ph.D. program. In addition, I want to thank Dr. Raj Kettimuthu (Argonne National Laboratory), Dr. Eun Sung Jung (Hongik University) and Dr. Nageswara S. V. Rao (Oak Ridge National Laboratory). They helped me navigate the complexities of high-performance data transfer tools and investigate their performance during my internship at Argonne National Laboratory. I also want to thank Andrew Hadenfeldt, Bryce Barnett and Steven Weaver for generously sharing their time and ideas during my work at ALLO Communications. Further, I want to thank Dr. Santosh Pitla (UNL) for his

inputs and feedback on the ERGO edge computing system.

I want to acknowledge Dr. Hesham Ali, Bhuvana Gopal, Dr. Nirnimesh Ghose, Dr. Abdul Salam, Shideh Yavary Mehr, Karthik Reddy Gorla, Natasha Pavlovikj, Jared Soundy and Boyang Hu for their help, advice and experience.

Finally, I'd be remiss if I didn't acknowledge the innumerable sacrifices made by my wife, Sneha. Her love, patience and nurturing kept me going while I pursued this terminal degree. I would also like to extend my deepest gratitude to my parents, Manjula and Anantha, for their unwavering support and guidance.

GRANT INFORMATION

This material is based upon work supported by the National Science Foundation under Grant Numbers OAC-1541442 and CNS-1817105. This work was completed using the Holland Computing Center of the University of Nebraska, which receives support from the Nebraska Research Initiative.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

The Internet today can be envisioned as a three-tier network. At the very top, we have applications generating massive amounts of information. The middle-tier is responsible for providing structure to the data from the application-layers using the internet protocol (IP) stack. Finally, the bottom-tier carries the information over a high-speed, distributed physical network. Applications exhibit diverse characteristics and place varying demands on network resources. Software defined networking (SDN) [1] and network functions virtualization (NFV) [2] along with advances in cloud computing makes building intelligent networks possible.

Typically, campus network infrastructures support multiple network traffic management goals. Commonly supported network traffic types include the following:

- Commodity internet traffic to support routine business tasks such as email and web
- High-performance networks for tasks associated with the scientific research process such as data retrieval, storage, and analysis from external sources
- Specialized (possibly separate) networks to meet specific project/workflow requirements like confidentiality, privacy, and anonymity.

These goals often impose conflicting requirements on network design and management. For example, networks designed for routine tasks such as email or web browsing are incapable of supporting large-scale data movement for data-intensive science. In this context, networks optimized and specially engineered for data-intensive tasks are necessary, without which, significant performance degradations would result. Further, many aspects of campus networks are hard to change without impacting normal network operation.

Increasingly, campus networks manage a multitude of large-scale data transfers. Big data plays a pivotal role in university research and impacts domains such as engineering, agriculture, natural sciences, and humanities. Over the years, numerous solutions have been proposed to manage and secure large-scale data transfers efficiently. Examples include optimized middlebox management, the inclusion of security policies at the network edge, and the Science Demilitarized Zone (Science DMZ). These solutions severely degrade data transfer performance or result in data flows completely bypassing the campus network management and security controls.

Campus networks, traditionally, are designed to handle a large number of small traffic flows and are ill-suited for high-volume science data transfers. Sharing campus network resources with scientific data transfers has many disadvantages. First, packet loss that is tolerated by campus local area networks (LAN) generally lead to severe performance degradations with bulk data transfers. Second, the wide area network (WAN) latency effects come into play reducing TCP performance. Lastly, campus network security controls optimize the security of small flows while sacrificing performance and therefore do not meet the stringent requirements for large data transfers over the WAN. SDN and NFV pave the way for novel traffic engineering methods that can overcome the limitations of traditional campus

networks. SDN and NFV architectures [3] are used successfully in big data environments for network management [4], QoS provisioning [5], and network design [6].

In this work, we study existing architectures for data-intensive applications, their implementation, strengths, and weaknesses. Next, we present SDN and NFV-based solutions for data-intensive applications. Our proposed software defined networking solutions span *network monitoring, network management, service differentiation, and security* for data-intensive applications.

1.1 Data-Intensive Science

Data-intensive science relies on massive amounts of data obtained from scientific studies (both experimental studies and simulation). Many science domains are data-intensive and data-driven. They rely on data acquisition, storage, and analysis for advancing fundamental research. Example data-intensive science domains are high energy physics (HEP), climate sciences, biology/genomics, combustion and light sources. In this section, we briefly summarize the big data challenges of different science domains.

High energy physics (HEP) is data-intensive, as advances in this domain require measuring the probability of "interesting events" in a large set of observations (typically 10^{16} or more particle collision observations in a year). The CMS (compact muon solenoid) [7] and ATLAS (A Toroidal LHC ApparatuS) [8] experiments at the large hadron collider (LHC) generate massive amounts of data ($\sim$ 10 petabytes per year). The ATLAS experiment currently stores over 100 petabytes of data and it is growing rapidly. The HEP community faces two important challenges namely: i) real-time data reduction, and ii) distributed data processing, analysis,

and distribution. The massive volume of analog data generated by the CMS and ATLAS detectors must be reduced in real-time to ensure that it can be stored at an acceptable cost. Furthermore, the data must be shared/distributed worldwide with scientists for further processing and analysis.

Climate science is another example of a data-intensive science domain that critically depends on large amounts of data available from heterogeneous sources worldwide. Sophisticated models of physical processes and interaction between realms (e.g., atmosphere, land, sea) are used by collaborative, multi-disciplinary teams to achieve scientific progress. Example data sources include advanced radiation measurement (ARM) sites and earth observing system (EOS) of satellites. The data from different sources is subject to (near) real-time processing and made available to numerous communities such as scientists, industry, policy-makers, etc.

Other data-intensive domains include the advanced photon source (APS) at Argonne national labs (ANL) and the linac coherent light source (LCLS) facility at SLAC National Accelerator Laboratory. Light sources are used by scientists from many domains ranging from material science to paleontology. Similar to the above domains, there is a strong need for collaborative and high-performance ecosystem at these light source facilities.

Popular services/frameworks for bulk data movement supporting scientific users include: i) The Globus Online service built with the GridFTP [9] protocol, and ii) the XROOTD [10] framework. These protocols and services integrate with data transfer nodes (DTNs) and high-speed parallel filesystems such as Lustre or GPFS. The DTNs are purpose-built for bulk WAN data transfers and typically have high-speed network interfaces (40/100 Gbps). In the following section, we describe some exemplary architectures for data-intensive science. These architectures integrate many networking components including SDN, virtual circuits, DTNs,

policy frameworks, etc. We also describe the architectural approach for securing high-volume transfers associated with data-intensive science.

1.2 Exemplary Data-Intensive Science Architectures

Many architectures have been proposed to support high-performance data transfers for science applications. These architectures try to meet both operational and security requirements while ensuring simplicity and scalability. Some example architectures are Science DMZ [11], SciPass [12], ScienceSDS [13], Medical Science DMZ [14], etc.

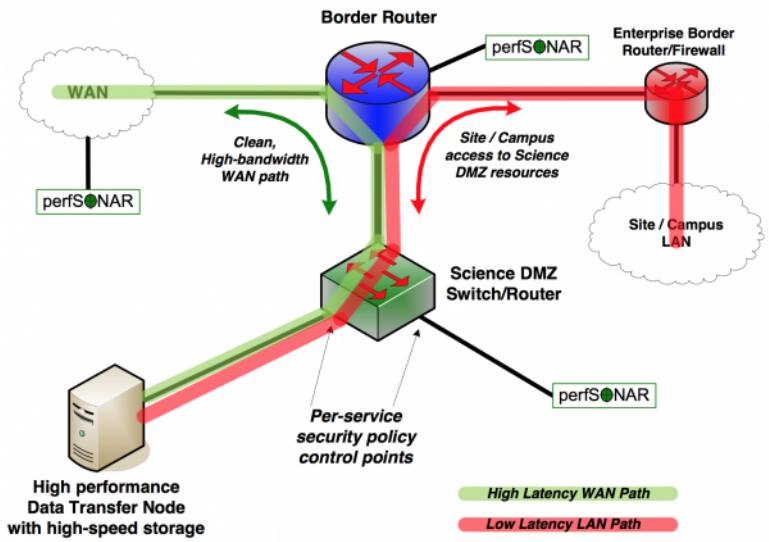


Figure 1.1: The Science DMZ Architecture (Source: ESNet).

The science DMZ [11] is a scalable network architecture for data-intensive science, built at or near an organization's local network perimeter. The architecture was developed by ESnet to address the network performance problems associated with high-volume, bulk data transfer needs of science applications. The Science DMZ integrates four key components including a network architecture distinct from general-purpose networks, a dedicated system for data transfers, a perfor-

mance measurement and network testing system, and policies and enforcement mechanisms for security. The architecture is shown in Figure 1.1. In this architecture, the DTN is connected directly to the border router. A crucial distinction of this approach is that the offsite data traverses only two devices, the border router, and the Science DMZ switch/router. The DTN is responsible for efficient WAN data movement, and access control lists enforce security policies for the DTN, without the need for a separate firewall.

SciPass [12] is a 100 Gbps capable SDN-based approach to Science DMZ. It combines an adaptive intrusion detection system (IDS) and dynamic traffic filtering to overcome the challenges posed by campus network firewalls. SciPass combines an SDN controller, a cluster of IDS sensors, a PerfSONAR host, and a firewall with a DTN to identify and filter flows. Policy definitions are used to determine whether flows can bypass the campus firewall controls.

Recently, a medical science DMZ was proposed by the authors in [14]. The proposed solution enables secure, high-volume data transfers of sensitive datasets between organizations while adhering to security and privacy regulations such as Health Insurance Portability and Accountability Act (HIPAA) [15]. The medical science DMZ re-engineers the science DMZ for use with restricted data by creating a secure, high-capacity, data-intensive portion within each organization. The trust relationships at each organization are used to ensure secure data movement to the appropriate computing facility.

Although these architectures provide adequate security and improve network performance for large-volume data transfers, they have some limitations. The science DMZ approach requires networks that are free of packet loss and stateful packet processing. While science DMZ performs selective monitoring and inspection of flows, it is not sufficient to detect all attacks. SciPass relies on IDS to

identify flows, and can, therefore, be ineffective when end-to-end data transfers are encrypted. In this dissertation, we develop novel solutions for monitoring, managing and securing large-scale data transfers at Holland Computing Center, University of Nebraska-Lincoln, a U.S. CMS Tier-2 site.

1.3 Application-Awareness

Existing architectures for securing data-intensive science rely on stateful packet processing techniques like deep packet inspection (DPI) or intrusion detection system (IDS) to classify packets/flows. These techniques are ineffective and place an undue burden on middleboxes due to the massive amounts of data transferred between end-points. Further, stateful packet processing can result in packet losses and increased detection latencies, thereby reducing the reliability of such techniques.

We take a unique approach to solving the above problems while minimizing the performance degradation effects of network middleboxes. In this dissertation, we develop an application-aware architecture to facilitate the secure exchange of application metadata with the network-layer. Our experience with deploying application-aware solutions for network monitoring, management and security have shown not only the benefits but also the reliability of our approach. Our novel application-awareness approach has been successfully employed in network monitoring, network management, and network security solutions for data-intensive science workflows. In the following, we outline a few networking research areas that have benefited from our application-awareness approach.

1.3.1 Secure and Reliable Flow Identification using Application-Awareness

As relying on stateful packet inspection for identifying large-volume flows is ineffective, we proposed SDN-managed network architecture for GridFTP transfers (SNAG) [16] (See Chapter 2). SNAG, unlike other approaches, relies on the exchange of application-layer metadata with the network-layer to provide flow intelligence to the network management system. Application-awareness, from the GridFTP servers in this case, is enabled by exposing a secure application program interface (API) from a trusted GridFTP process. The GridFTP server automatically forwards application metadata to the SDN controller. All GridFTP connection metadata including source/destination addresses, port pairs, session information, etc., are used by the network layer to identify and classify all active flows reliably. This flow identification and classification is vital to the network operator to ensure that appropriate security policies and enforcement mechanisms are defined.

1.3.2 Large-scale Data Transfer Monitoring, Management and Service Differentiation

SNAG, using an application-aware approach to traffic classification, helped us develop a unique monitoring system for data-intensive science. Our SNAG-enabled monitoring system can monitor, in real-time, CMS and LIGO (laser interferometer gravitational-wave observatory) flows. Application-layer metadata from the GridFTP servers are used to create fine-grained monitoring views. These views provide additional insights into data-intensive science transfers such as user/project, workflow, and file statistics. The monitoring views created using application-awareness *cannot* be obtained using traditional methods that use stateful packet processing. This is due to the encrypted nature of GridFTP connections. At the

Holland Computing Center (HCC), UNL, we have used SNAG to classify over *1.5 billion* connections successfully.

Further, we have used the traffic classification information provided by SNAG to achieve service differentiation [17]. For example, flows identified by SNAG can be individually subject to different quality of service (QoS) policies. Through the secure exchange of application metadata with the network layer, intelligent and adaptive network management decision-making is possible. Also, we can exploit application-layer metadata to define security policies for application flows more accurately.

1.3.3 An SDN/NFV Approach to Optimized Service Chain Mapping and Securing Data-Intensive Science Transfers

Network Function Virtualization (NFV) brings a new set of challenges when deploying virtualized services on commercial-off-the-shelf (COTS) hardware. We can dynamically manage network functions to provide the necessary services on-demand. Further, services can be chained together to form a larger composite. Application-awareness is beneficial when mapping service function chains (SFCs) across different data centers with the objective of reducing the flow processing costs. The SFC mapping problem is critical to enhancing the virtualized service networks' performance, as it places high demands on the performance of these service functions. Application-aware architectures can also secure data-intensive science workflows. Using the traffic classification information, we developed ScienceSDS [13]. ScienceSDS is a software-defined security framework that employs a virtualized security services infrastructure to provide security-as-a-service for data-intensive science. Here, we use the traffic classification information to redirect only flows of interest intelligently to virtualized security services. Unlike traditional

middleboxes and security appliances, our approach using virtualized services is highly scalable. The SNAG application-aware approach also forms an important component of our GridFTP transfer anomaly detection system proposed in the work in [18].

1.3.4 Predictive and Intelligent Load Balancing using Application-Awareness

Application-layer information is valuable when developing intelligent load-balancing solutions to improve server utilization. We can combine application-aware SDN with machine learning approaches to model large data transfers in the campus network. A multi-layer deep learning network can automatically discover data representations that can then be used to infer information from the dataset without the need for complex analysis. We can successfully employ application-aware architectures with SDN to model and forecast network information using representational learning techniques such as machine learning and deep learning. The resulting predictors can aid in network resource allocation. Load balancing forms a critical component of big data network architectures as they directly influence application response times and maximize throughput via optimized traffic delivery to the application servers. Large-volume data transfers associated with big data provides many opportunities for understanding usage patterns and gain insights into network resource requirements. Rather than viewing big data systems as placing an undue burden on campus networks, we can exploit the insights gained in better understanding user/traffic demands. This results in optimized resource allocation to better serve the needs of campus network users.

1.3.5 Optimized Service Delivery in Distributed Service Mesh Architectures

Cloud computing has led to the development of new paradigms for application deployments and service delivery. Service instantiation, deployment and rapid service delivery in cloud native environments increasingly rely on microservice architectures. To take full advantage of cloud native microservice architectures automated microservice deployment and orchestration systems that optimize service placement are necessary. Such orchestration systems should carefully map the available network resources with the deployed services to ensure optimum end-to-end service performance. Therefore, a critical challenge is to ensure reliable service-to-service communication across distributed infrastructures. Network service meshes have emerged as a useful pattern to enable inter-service communication by providing an addressable infrastructure layer. Service meshes can also provide several advanced features including declarative control over service behavior using a policy-based configuration, service traffic visibility, service resilience, service discovery and security. While container orchestrators focus on workload management, scheduling, health monitoring and discovery, a service mesh focuses on mitigating unmet service-level requirements. Service meshes ensure secure service-to-service communication and service discovery across distributed infrastructures using a network of sidecar containers or proxies. These sidecars/proxies form the service mesh data plane and are responsible for relaying traffic securely between services and client endpoints. Further, they are combined with ingress (reverse proxy) and egress (forward proxy) gateways to control inbound and outbound traffic, respectively.

1.4 Contributions

The research presented in this dissertation focuses mainly on the broad topic of application-layer and network-layer collaboration for network monitoring, management and security. Application-layer and network-layer collaboration, referred to as "application-awareness," is the exchange of application-layer metadata with the network-layer. We can optimize network-layer functionality and decision-making by using real-time information about the applications that connect to it. As academic campus networks support large-scale data transfers for data-intensive science, application-awareness is particularly useful to securely identify, manage, and monitor traffic flows.

Large-scale data transfer workflows for data-intensive science rely on high-performance, scalable, and reliable protocols for moving large amounts of data over a high-bandwidth, high-latency network. GridFTP is a widely used protocol for wide area network (WAN) data movement. However, as the GridFTP protocol does not share connection information with the network-layer, network operators have reduced flexibility, particularly in identifying/managing flows across the network. In **Chapter** 2, we address this problem by deploying a production "*application-aware*" software defined network (SDN) for managing GridFTP transfers for data-intensive science workflows. We first propose a novel application-aware architecture called SNAG (SDN-managed Network Architecture for GridFTP transfers). SNAG combines application-awareness with SDN-enabled network management to classify, monitor and to manage network resources actively. Until now, our SNAG deployment has successfully classified over *1.5 Billion* GridFTP connections at the Holland Computing Center (HCC), University of Nebraska-Lincoln (UNL). Next, we develop an application-aware SDN system to provide differentiated network services

for distributed computing workflows. At HCC, we also demonstrate how our system ensures the quality of service (QoS) for high-throughput workflows such as Compact Muon Solenoid (CMS) and Laser Interferometer Gravitational-Wave Observatory (LIGO). Further, we also demonstrate how application-aware SDN can be exploited to create *policy-driven* approaches to achieve accurate resource accounting for each workflow. We present strategies for implementing differentiated network services and discuss their capacity improvement benefits. Lastly, we provide some guidelines and recommendations for developing application-aware SDN architectures for general-purpose applications. This work was published in the Third Workshop on Innovating the Network for Data-Intensive (INDIS) at Supercomputing 2016 [16], and at the 2017 IEEE International Conference on Advanced Networks and Telecommunications Systems (ANTS) [17]. An extended version of this work has been submitted to the IEEE/ACM Transactions on Networking and is currently under review.

In **Chapter** 3, we address an important problem of mapping service function chains (SFCs) across different data centers with the objective of reducing the flow processing costs. The SFC mapping problem is critical to enhancing the virtualized service networks' performance, as it places high demands on the performance of these service functions. First, we develop an integer linear programming (ILP) formulation to optimally map service function chains to multiple data centers while adhering to the data center's capacity constraints. Next, we propose a novel application-aware flow reduction (AAFR) algorithm to simplify the ILP to significantly reduce the number of flows processed by the SFCs. We study the SFC mapping problem for multiple data centers and evaluate the performance of our proposed approach with respect to three parameters: i) impact of number of SFCs and SFC length on flow processing cost, ii) capacitated/uncapacitated

flow processing cost gains, and iii) balancing flow-to-SFC mappings across data centers. This work has been published in the 2018 4th IEEE Conference on Network Softwarization (NetSoft) [19]. An extended version of this work has been submitted to the Elsevier Computer Networks Journal and is currently under review.

In **Chapter** 4, we propose an application-aware intelligent load balancing system for high-throughput, distributed computing, and data-intensive science workflows. We leverage emerging deep learning techniques for time-series modeling to develop an application-aware predictive analytics system for accurately forecasting GridFTP connection loads. Our solution integrates with a major U.S. CMS Tier-2 site; we use a real-world dataset representing *800 million* GridFTP transfer connections measured over 2 years to drive our predictive analytics solution. First, we perform extensive analysis on this dataset and use the connection loads as an example to study the temporal dependencies between various user roles and workflow memberships. We use the analysis to motivate the design of univariate and multivariate deep recurrent neural network (RNN) models for understanding the long-term temporal dependencies and predicting connection loads. We explore two RNN techniques for GridFTP load forecasting, namely long short-term memory (LSTM) and gated recurrent unit (GRU) based deep learning networks. We develop a novel application-aware, predictive and intelligent load balancer, APRIL, that effectively integrates application metadata and load forecast information to maximize server utilization. We conduct extensive experiments to evaluate the performance of our deep RNN predictive analytics system and compare it with other approaches such as ARIMA and multi-layer perceptron (MLP) predictors. Lastly, we also demonstrate the effectiveness of APRIL by comparing it with the load balancing capabilities of an existing production Linux Virtual Server (LVS) cluster. This work has been published in the 2018 IEEE International Conference

on Advanced Networks and Telecommunications Systems (ANTS) [20] and the IEEE INFOCOM 2019 - IEEE Conference on Computer Communications [21]. An extended version of this work has been submitted to the IEEE Transactions on Network and Service Management is currently under review.

Chapter 5 proposes a scalable edge computing architecture, ERGO, for environments characterized by limited and intermittent connectivity. We demonstrate ERGOs' effectiveness in an agricultural IoT setting, present an exemplary image processing application, and examine its computational and service requirements. We design ERGO for connectivity-challenged environments with limited (possibly periodic) or no wide area network (WAN) connectivity. ERGO also provides representational state transfer (REST) application programming interfaces (APIs) for various management and end-user tasks. Therefore, ERGO combines container orchestration systems with web-services APIs to provide scalable, edge-enabled services. Our implementation and extensive evaluations show that ERGO can operate independently of cloud-backed assistance, is highly scalable, modular, and affords composability benefits to Ag-IoT systems. Next, we present a generalized service performance measure (GSPM) for multi-cluster distributed service mesh architectures. As service mesh architectures gain popularity, optimized service-to-service communication is vital for ensuring quantifiable end-to-end service performance. Real-world service mesh deployments often employ performance metrics that exhibit large variability across clusters. However, service traffic forwarding decisions in a service mesh rarely rely on localized (i.e., in-cluster) performance metrics, thereby skewing routing decisions in favor of latency rather than overall service performance. To solve this problem, we propose a generalized service performance measure (GSPM) that provides a metric- centric view of service performance. First, we present the average service distance (ASD) measure

and analyze its performance by modeling two prominent service mesh structures. Next, we develop GSPM by augmenting the average service distance with localized metrics that are representative of the service/workload performance. Lastly, we demonstrate the effectiveness of our proposed GSPM-based dynamic routing approach by evaluating it on a real-world service mesh testbed. Our preliminary work on ERGO was presented at the 3rd USENIX Workshop on Hot Topics in Edge Computing (HotEdge 20) [22]. This work appears in the 2021 IEEE LANMAN conference [23].

1.5 Organization of the Dissertation

This dissertation is organized as follows: First, we present our novel application-aware architecture, SNAG, and its applications to SDN-based network monitoring, management and service differentiation in Chapter 2. In Chapter 3, we develop an integer linear programming (ILP) formulation to optimally map service function chains (SFC) to multiple data centers. We also present a novel application-aware flow reduction (AAFR) technique to reduce the SFC flow processing costs. Next, in Chapter 4, we propose APRIL, an application-aware intelligent load balancing solution for high-throughput, distributed computing, and data-intensive science workflows. APRIL effectively integrates application metadata and deep learning predictors to forecast GridFTP server loads and maximize server utilization. In Chapter 5, we propose ERGO, a scalable edge computing architecture for agricultural IoT environments. We also present a comprehensive model for network service meshes and develop an average service distance measure for two dominant service mesh structures. Further, we develop a generalized service performance measure to quantify and evaluate acceptable end-to-end service performance

bounds. Lastly, we conclude our work and describe the future work in Chapter 6.

Chapter 2

SNAG: SDN-managed Network Architecture for GridFTP transfers using Application-Awareness

2.1 Introduction

Data-Intensive science workflows transfer large amounts of data and are often characterized by high-throughput distributed computing workflows. Examples of such workflows include data transfers from the Compact Muon Solenoid (CMS) [7], a particle physics detector at CERN, and the Laser Interferometer Gravitational-Wave Observatory (LIGO) [24] experiment developed for detecting cosmic gravitational waves. These workflows depend heavily on high-speed networking and high-performance computing/storage for analysis tasks. Dedicated high-speed networks including the Energy Sciences Network (ESnet) and the Internet2 infrastructure provide the necessary network resources and generalized frameworks for these workflows. Recent advances in SDN and OpenFlow [25] allow network applications to create and manage fine-grained network control policies. Such fine-grained control can influence policy-based decision-making for traffic management/monitoring, differentiated network services, and security. Although using dynamic resource allocation (e.g. OSCARS [26]) or software defined networking can improve the end-to-end data transfer performance of scientific workflows,

numerous problems exist with resource management and providing differentiated services in the campus network. The HCC at UNL is a U.S. CMS Tier-2 site (henceforth referred to as the "*site*") performing frequent CMS transfers to Fermilab. In this work, we present an application-aware SDN and a policy-driven approach to classifying, monitoring and actively managing network resources for data-intensive science projects at the experiment level while allowing for accurate accounting of site-wide resources.

We propose SNAG (SDN-managed Network Architecture for GridFTP transfers), a novel application-aware SDN architecture for managing GridFTP transfers. We present a detailed description of SNAG and combine application- and network-layer collaboration with SDN-enabled network management to monitor, manage, and account for network resources used by GridFTP transfers. We expose an Application Program Interface (API) between the trusted GridFTP process and the network layer, allowing the network to accurately and reliably track flows via application metadata. We then develop a differentiated network services solution (site-level) using SNAG to provide quality of service (QoS) to flows from different data-intensive science workflows at HCC. We also demonstrate a policy-driven approach to network monitoring, management, QoS, and accounting for data-intensive science workflows at HCC.

2.1.1 Contributions and Organization

Our contributions are outlined below:

1. *Site-wide GridFTP traffic classification and monitoring*: We deploy an SDN-based network architecture to effectively and securely classify/monitor GridFTP traffic and describe SNAG implementation on our project testbed.

2. *Monitoring views from Application- and Network-layer collaboration*: We demonstrate how SNAG exploits application- and network-layer coordination to create unique and real-time network monitoring views. These views *cannot* be achieved using traditional networking techniques due to the encrypted nature of the transfers.

3. *Application-aware QoS*: We develop an application-aware approach to provide differentiated network services and QoS, thus enabling active network management and traffic prioritization. Moreover, we achieve greater control over data-transfer workflows where priorities are applied to individual users and/or experiments. Using SNAG, service differentiation and QoS is applied to network traffic based on application-layer metadata.

4. *Policy-driven network management*: By defining per-flow policies, we develop a policy-driven approach to network management. We demonstrate how this approach can handle both opportunistic and scheduled data transfers at HCC.

5. We demonstrate the scalability of our solution by deploying SNAG on a project testbed network at HCC that has been set up to integrate with a U.S. CMS Tier-2 site.

This chapter is organized as follows: Section 2.2 presents the motivation and the related work; Section 2.3 provides the overall context of this work within the SDN field for experimental science data transfers and provides a brief introduction to application-aware SDN; Section 2.4 presents a detailed description of the SNAG architecture and its implementation; In Section 2.5, we present our experimental setup, testbed topology, and our data management framework. In Section 2.6, we

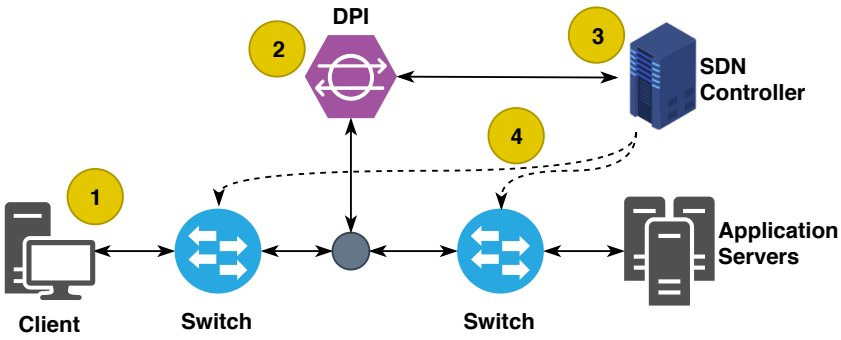
present the *traffic classification* use case using the SNAG architecture; Section 2.7 presents the *network monitoring* use case, with monitoring views and forecasting capabilities of SNAG; Section 2.8 presents the *differentiated network services* use case and active network management for science data transfers using SNAG; In Section 2.9, we provide some recommendations for building application-aware architectures; Lastly, in Section 2.10, we conclude our work.

2.2 Motivations and Related Work

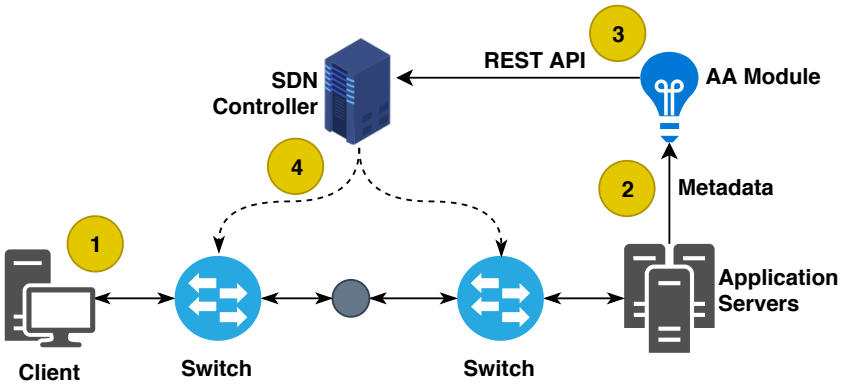
The GridFTP protocol [27, 28] has become a widely used network protocol for WAN data movement in cluster/grid environments. Data-intensive science workflows rely on distributed high-throughput computing systems for experimentation and analysis tasks. The CMS computing model [7] is employed by workflows such as CMS or LIGO on the Open Science Grid (OSG) [29] for moving large scientific datasets across different computing sites using the GridFTP protocol. The GridFTP protocol overcomes the well-known limitations of transmission control protocol (TCP) for use in R&E networks that are characterized by high latency and high bandwidth. This approach, at the cost of fairness, maximizes throughput for large-scale data movement. Further, GridFTP maximize data transfer throughput by creating parallel TCP sessions for each connection. At HCC, both CMS and LIGO experiments rely on the GridFTP protocol for WAN data movement.

The GridFTP protocol breaks TCP fairness by focusing on end-to-end data transfer throughput maximization by creating parallel TCP sessions for each established connection. Due to this approach, the use of traditional network management and traffic prioritization techniques with GridFTP are becoming increasingly limited. As an example, an operator may be required to assign different traffic priorities to

users of different workflow memberships within the same experiment. Therefore, the network operator, in order to apply appropriate actions, must be provided with an efficient and suitable policy-based mechanism to accurately differentiate flows based on user, experiment, and workflow membership information. During connection establishment, as GridFTP relies on the use of an encrypted control channel, traditional techniques for traffic classification that employ packet inspection are ineffective as packet sniffing does not help the network classify flows. The inability to obtain accurate traffic classification information means that prioritizing flows based on either workflow membership, or based on users within a workflow is not possible. As an example, a low-priority user may initiate thousands of TCP streams in comparison to a high-priority user. Different experiments utilize the same network source and destination addresses, preventing network-based segmentation or prioritization of traffic. Alternately, *within* an organization, there is a need to differentiate high-priority transfers versus test transfers. Application-awareness (see Section 2.3), through application and network-layer collaboration, is essential to obtain accurate information about current and ongoing transfers. By using application metadata to identify flows, we can overcome the above mentioned limitations including the ones arising from the encrypted nature of the traffic. Although numerous works propose the use of overlay networks, separate session adaptation layers for the network control plane, and network middleboxes for end-to-end data transfers, the use of application-aware architectures resulting in application- and network-layer collaboration is limited. In the following sections, we provide a brief overview of works that share some common aspects with our application-aware networking approach.



(a) Stateful Processing.



(b) Application-awareness.

Figure 2.1: Stateful packet processing vs. application-aware approaches.

2.2.1 Network Management for Data-Intensive Science Transfers

An early effort to create an infrastructure layer based on session adaptation layer was proposed in Phoebus [30, 31]. It relied on the strategic placement of "Phoebus Gateways" (PGs) to dynamically allocate network resources using segment-specific transport protocol adaptations. The traffic between end-points would then be rerouted to the PGs for segment specific optimizations. OSCARS [26] proposed the use of an overlay network architecture for enabling high-throughput, coordinated data transfers over a WAN by leveraging the PGs. While Phoebus provides the routing infrastructure for optimal path selections, OSCARS allows for high-bandwidth network segment provisioning. Other approaches such as [32] and

[33] overcome the limitations of typical scientific workflow communication tasks requiring file transfers by relying on data locality in cloud storage services. Works such as [34, 35] explore the application of differentiated scheduling and network parallelism to achieve high WAN data transfer performance over lossless dedicated resource provisioning systems such as ESnet and Internet2. The authors in [34] propose the use of multiple paths on the WAN for the same application transfer session. Using a bi-objective file transfer scheduling approach to reduce latencies, they show that resource reservations are not necessary to meet the requirements of response-critical WAN applications.

2.2.2 SDN-enabled Network Management and Monitoring

The eXtensible Session Protocol (XSP) [36] proposed a flexible protocol architecture using the notion of a "session" to manage network states and provide application services. The "session" manages data contexts, forwarding rules, state information, connection lifetimes, or any other application interaction information. Applications use this information to signal the network and change its configuration to suit application needs. An XSP daemon (XSPd) containing a number of protocol specific backends such as OSCARS [26], Terapaths [37, 38], OpenFlow [25] etc., allows the application to signal the underlying provisioning service and receive feedback. However, unlike our proposed solution, XSP provides limited application-layer information and is loosely integrated with the network-layer. Other solutions have been proposed to create both active and passive network monitoring systems using SDN/OpenFlow including OpenTM [39], OpenNetMon [40], and OFMon [41]. These approaches have the advantage of maintaining simplicity in the network core. However, their ability to accurately classify/monitor traffic *at scale* with high volume transfers is untested.

The majority of the WAN transfers at HCC do not use resource reservations, as bottlenecks in the campus network are created primarily from resource sharing across different workflows. Instead, we focus on site-level improvements and intelligent decision making at the network edge to enhance transfer performance. Unlike the network management architectures discussed in Section 2.2, our work on SNAG focuses on accurate classification of flows at both the site-level (i.e. campus network) and the experiment-level. We focus on optimizing data transfers for users within a project membership and also between experiments. These optimizations can, however, be combined with any of the above mentioned WAN transfer strategies to further enhance end-to-end data movement. By focusing on site-level (i.e. campus network) improvements rather than WAN performance improvements, SNAG provides maximum flexibility to the site operator to prioritize traffic and network resources for each experiment. By bridging the gap between application-layer information and the corresponding network-layer flows, our work integrates GridFTP with SDN to provide policy-based mechanisms for network traffic management. SNAG focuses on ensuring accurate traffic classification and network flow management at a single site. It does not require changes to the the entire network but only changes at the site-level. Existing solutions do not provide fine-grained controls at the end-points to prioritize specific flows but focus solely on large-scale end-to-end data transfer throughput improvements. These approaches improve the end-to-end GridFTP transfer performance through multi-path bulk data movement by employing XSP, GridFTP striping and data-path parallelism. Different from these approaches, SNAG focuses on GridFTP flow classification, network monitoring and differentiated network services through active network management at the site-level. This allows for optimizations in resource allocation, site-specific policy enforcement, resource-usage accounting

and traffic prioritization.

2.3 Application-Aware SDN

Applications designed for data-intensive science interact with massive amounts of data and place varying demands on the underlying network infrastructure. Applications also exhibit diverse characteristics that impact the resource availability, quality of service (QoS), latency and network security. Although networks can use application-layer metadata to understand an application's resource requirements and make intelligent decisions, such exchange is often *limited or non-existent*.

Application-awareness is the exchange of application-layer metadata with the network, resulting in application- and network-layer collaboration. Thus, application-awareness can provide intelligence for network-layer decision-making. Through application-awareness, a network can optimize its functionality by obtaining real-time information about the applications that connect to it. This approach is different from logging application-layer information for use with monitoring and analysis purposes in a number of ways. Firstly, application-awareness provides real-time feedback about resource requirements and application/connection states. Through application-awareness, a network-layer entity can securely interact with an application process to adapt to its requirements. Secondly, log systems (i.e. events or transactions) are built with the intention of providing an audit trail for future analysis, and are therefore designed to record events of interest on occurrence. However, unlike logging, the application-aware approach exposes APIs to securely communicate all application-layer metadata to optimize network-layer functionality. Lastly, application-awareness can be employed to securely exchange application metadata with a network-layer controller even if the application uses

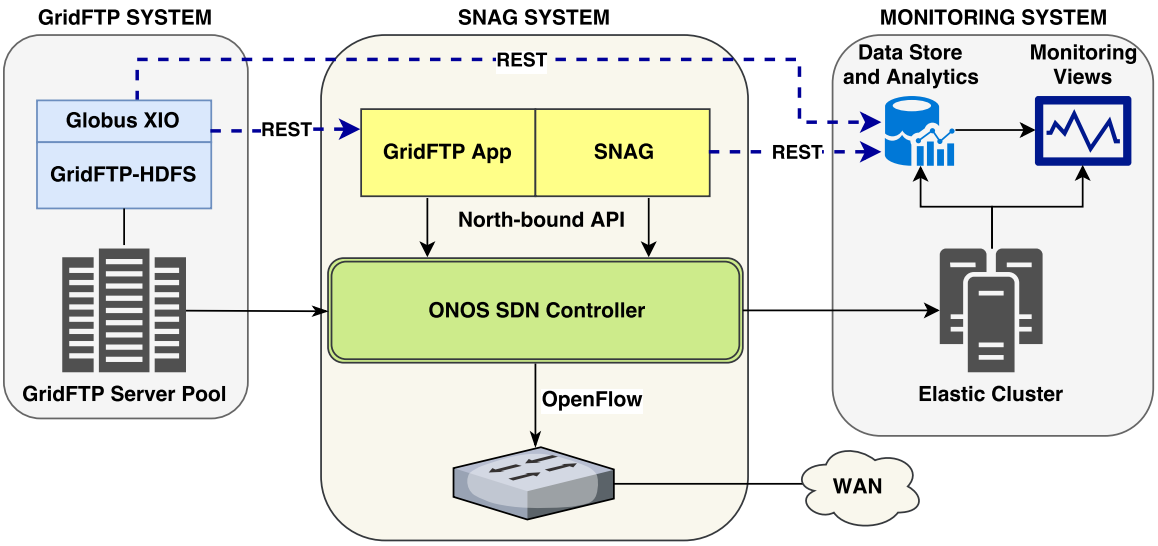


Figure 2.2: SNAG architecture consisting of the GridFTP, SNAG, and monitoring components.

an encrypted transport-layer connection between endpoints. Therefore, in order to obtain application-metadata for decision making, the network-layer need not resort to external stateful packet processing techniques like deep packet inspection (DPI). We note that the application-awareness approach is distinct from network-aware applications, which involve application-specific network topology optimizations.

In Figure 2.1, we present an example of stateful packet processing and how it differs fundamentally from application-aware approaches. Figure 2.1a shows the process of extracting application-layer metadata using stateful packet processing. The client first establishes a connection with the application server(s) and initiates a data transfer. A stateful packet processing system such as a deep packet inspection (DPI) unit is responsible for inspecting packet headers from the underlying traffic and infer application metadata parameters. This information is communicated to an SDN controller that uses the application-layer metadata in decision making and corresponding rule/policy enforcement in the data plane. However, when the

control/data channel communication between the client and application server is encrypted end-to-end, DPI or other stateful packet processing systems are severely challenged in their ability to extract application-layer information. This limitation is further compounded by the large volumes of data transfers that are typical of data-intensive science.

An application-aware approach is shown in Figure 2.1b. In contrast to stateful packet processing, application-aware architectures do not rely on external packet processors to obtain application metadata. Instead, when a client establishes a connection with the application servers, the application servers securely communicate the connection parameters and associated application metadata directly to the SDN controller through a secure channel. Thus, our approach provides a secure exchange of accurate application metadata with the network layer to enhance network management decisions. Further, as our approach is unencumbered by stateful packet processing units, they are highly scalable and suitable for data-intensive applications. Next, we present the SNAG architecture and its integration with the GridFTP application servers.

2.4 SNAG Architecture

GridFTP transfers at HCC are characterized by high data transfer rates and consume significant network resources. To overcome the fairness limitations of TCP, each GridFTP connection instantiates multiple TCP streams, and in practice, it is common for each connection to set up 10 or more streams frequently. We also note that each stream can interact with a different storage system during a data transfer operation. At HCC, for a given use-case, we have also observed over 10,000 parallel TCP streams for data transfer. As a data transfer at HCC depends on both the

successful authentication of users and availability of data at the destination storage servers, managing users' network resources while ensuring availability has been historically problematic. Both data availability information at the destination storage servers and user authentication information are application-layer information. In this context, typical network QoS techniques that rely on Layer-2 information are ineffective as both low- and high-priority data transfers may use the same source and destination endpoints. In our work, we extend the Globus GridFTP server to integrate with an ONOS [42] SDN controller. This approach ensures that the application-layer information is made available to the SDN controller for making network-layer decisions and also to provide differentiated network services. Thus using SNAG, the SDN controller obtains all necessary application metadata for network-layer decision-making including connection strings, user/file information, storage directories, etc. for all ongoing data transfers. Our proposed solution integrates the ONOS SDN controller framework (using the north-bound API) with an extended GridFTP application server module designed to obtain application metadata for network management tasks. In this section, we describe our implementation architecture and how the extended GridFTP application servers and the ONOS SDN controllers interact with SNAG to achieve application-awareness.

2.4.1 Implementation

SNAG integrates the GridFTP application servers with network-layer SDN control using application-awareness. Our SNAG architecture is tested on the ONOS [42] Magpie release (1.12.0). ONOS is an open-source community SDN software framework which provides a programmatic control plane. Next, we share our experiences with the deployment of SNAG at HCC and how application-awareness can be used for monitoring and managing data transfers securely.

SNAG enables the mapping of application-layer metadata (GridFTP connection information) about network flows to the ONOS SDN controller using application-awareness. This mapping can be used to not only monitor flows from different user/workflows, but also to provide QoS and differentiated services at a fine-grained level. Figure 2.2 shows the interactions between the three main components of SNAG, namely:

1. GridFTP application servers, the Globus eXtensible Input Output (XIO) Module, and the Hadoop Distributed File System (GridFTP-HDFS) plugin.

2. SNAG system with the ONOS SDN controller and two SDN applications namely: i) ONOS GridFTP application, and ii) ONOS SNAG application.

3. An integrated monitoring system based on the Elastic stack [43], with Elasticsearch as a data store and Kibana for visualization.

At HCC, we deploy Hadoop Distributed File System (HDFS) [44] as the GridFTP servers' storage layer as this file system has fault-tolerance built-in natively. The GridFTP-HDFS plugin is used by the GridFTP application servers to interact with the HDFS storage infrastructure. As the plugin can access the file system layer, it can retrieve the GridFTP application metadata associated with ongoing data transfers. The Globus eXtensible Input Output (XIO) module exposes RESTful APIs to securely communicate the GridFTP application metadata to the SDN controller. The SDN controller, through a separate SDN-side application (called the ONOS GridFTP application), can use the APIs to securely query the GridFTP servers about new and ongoing data transfers. The XIO module also allows us to extend the functionality of the GridFTP application servers easily. Using the APIs, the ONOS GridFTP application retrieves various connection parameters including: i) GridFTP

client/server connection strings (src/dst IPs and port-pairs); ii) User information and associated workflow/experiment memberships; iii) Filenames; iv) Transfer direction (upstream or downstream); v) Transfer event information corresponding to one of STARTUP, UPDATE and SHUTDOWN; and vi) Connection timestamps.

This information corresponds to a *nonuple* (i.e. 9-tuple) and is used in traffic classification, monitoring/forecasting, and for active network management. The Globus XIO callout module initiates a RESTful API call for every transfer and sends the above information to the ONOS GridFTP application. The ONOS GridFTP application interfaces with the SNAG application (SDN-side) to preprocess the *nonuple*. SNAG then coordinates both passive network monitoring and active network management tasks. Thus, SNAG can be used to manage GridFTP transfers by configuring the appropriate flow rules through ONOS. The SNAG RESTful APIs are described below:

- ONOS GridFTP application includes a set of APIs to retrieve the nonuple information. The APIs provided are GET, POST, and DELETE.
- ONOS SNAG application for connection information preprocessing and flow management tasks. The APIs provided are GET, POST, PUT, and DELETE.

These APIs are used by the SDN-based network management system differentiate between GridFTP and non-GridFTP flows using application-layer information. Further, flow-specific treatment can be applied to either flows by applying the appropriate forwarding/routing policies. Through this approach, we use SNAG to analyze the flows and monitor network performance (see Section 2.7) and provide differentiated network services based on pre-configured network policies (see Section 2.8). SNAG can also be used in other scenarios, for example, based on the application-level user/workflow membership information, some trusted

user/workflow traffic can be routed to bypass stateful packet processing systems (e.g. ScienceSDS [45]) such as firewalls, DPIs etc. Such an approach can reduce the burden on the stateful packet processing systems while improving the data transfer performance.

2.5 Experimental Setup and Testbed

In this section, we present our experimental setup, integration with the GridFTP servers to enable application-awareness, the testbed network topology and its capabilities, and our Elastic cluster data store.

2.5.1 Integration with GridFTP

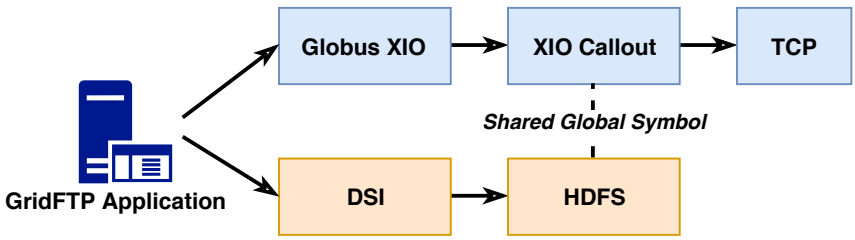


Figure 2.3: XIO and GridFTP interaction.

The Globus eXtensible Input Output (XIO) [46] library provides a single user API for all Grid IO protocols with the Globus Toolkit [47]. The XIO is a pluggable module with open/close/read/write API operations. SNAG uses an XIO Callout (see Section 2.4.1) to interface directly with the HDFS [44] storage layers using the GridFTP-HDFS plugin. The Globus XIO is an efficient and highly optimized IO library that is used by SNAG to create a scalable interface for retrieving GridFTP application metadata. The XIO module and its interaction with the GridFTP servers is shown in Figure 2.3.

2.5.2 Network Testbed Topology

The Holland Computing Center (HCC) at the University of Nebraska-Lincoln is a U.S. CMS Tier-2 site. HCC frequently handles high-throughput and high-priority data transfers from the Compact Muon Solenoid experiment to Fermilab. It also handles low-priority data transfers that are opportunistic in nature to the same destination. HCC currently manages over 3 petabytes of data from the CMS experiment and over 40 terabytes data from the LIGo experiment. Data transfers at HCC widely use the GridFTP protocol for bulk batch transfer jobs while interactive jobs are transferred over the XROOTD [10] file transfer protocol. Our SNAG architecture effectively combines several essential components such as:

- An ONOS SDN controller for programmable network management and traffic forwarding/routing tasks.
- An ONOS SNAG application for connection information preprocessing and flow differentiation/management tasks.
- GridFTP application servers interacting with the HDFS storage layers using the GridFTP-HDFS plugin.
- An XIO Callout developed for both retrieving the GridFTP application metadata securely and for communicating the metadata to the ONOS GridFTP application.
- 100 Gbps WAN connectivity linking HCC to the Internet2 infrastructure.

A Brocade MLXe IP/MPLS router placed at the data center border connects to the WAN over a 100 Gbps link as shown in Figure 2.4. Within the HCC network, a 40 GbE Dell S6000 switch serves the CMS cluster network core. This switch also

hosts the production GridFTP and XROOTD servers as shown in Figure 2.4. On the same network, a test network is also created which consists of a GridFTP server removed from the production pool along with an OpenFlow enabled Edge-Core AS4600-54T switch.

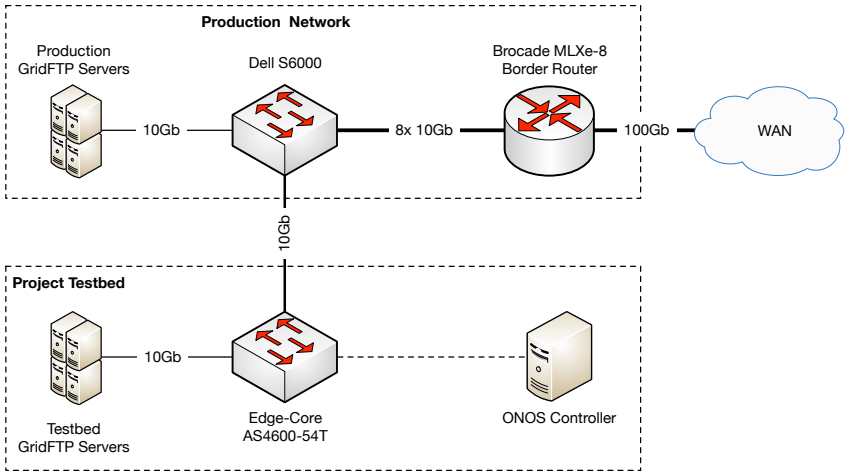


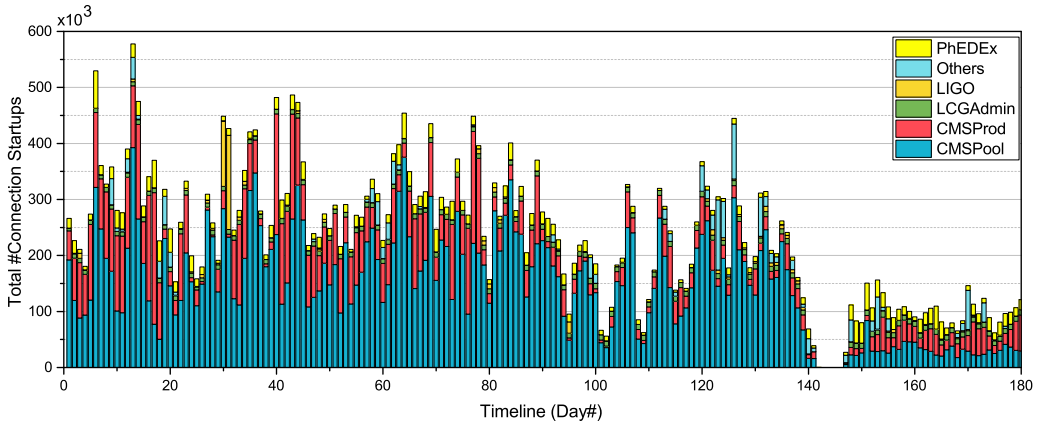
Figure 2.4: GridFTP and SDN testbed network topology with external (WAN) connectivity.

2.5.3 Elastic Cluster Data Store

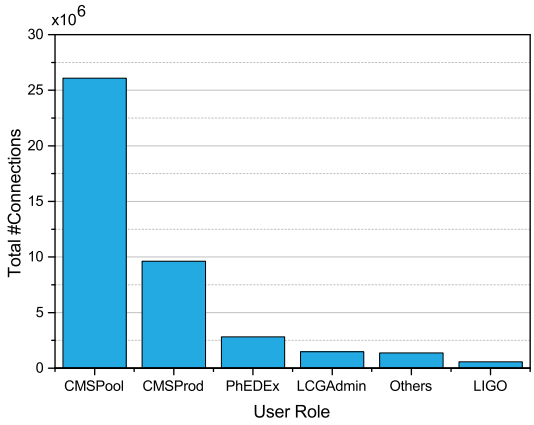
The application-aware information obtained using SNAG is sent to an Elastic stack [43] cluster for storage, monitoring and analysis. We use *Elasticsearch* for monitoring data indexing, processing and analysis. Real-time monitoring views are created using the *Kibana* visualization tool built into the Elastic stack. Our Elastic cluster is composed of 11 nodes and also serves as our monitoring system. Our Elastic stack cluster configuration is as follows:

- 3×Dell SC1435, 16GB RAM, 250GB HDDs for the master nodes.
- 3×Sun SunFire X2200, 32GB RAM, 240GB SSDs for hot data or ingest nodes.
- 5×Sun SunFire X2200, 32GB RAM, 2TB HDDs for warm data nodes.

We note that 1 Gb Ethernet connections are used to connect all of the above nodes. All application-layer metadata is sent to the Elastic cluster using a *syslog* style file. This file feeds a *filebeat* agent (a lightweight data shipper for the Elastic stack) which in turn feeds the *logstash* system, a server-side data ingestion pipeline on the Elastic cluster.

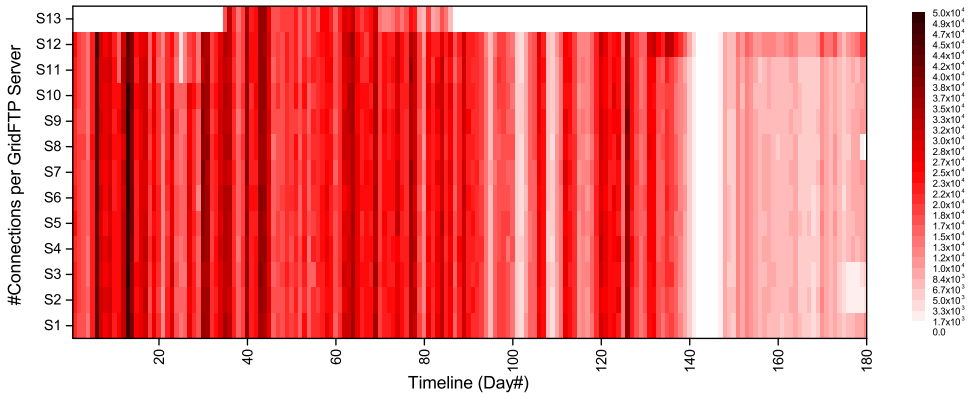


(a) Total number of connections (STARTUP events) per day classified by user type measured over a 6-month period.

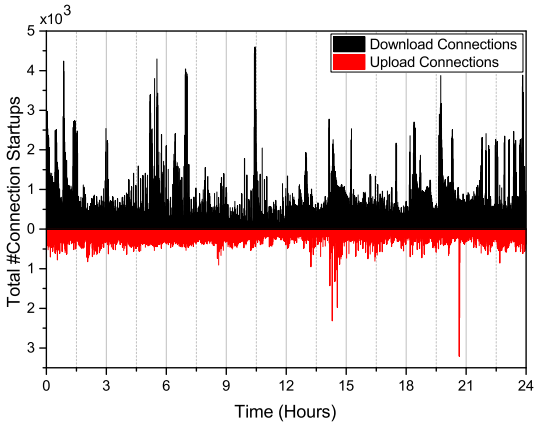


(b) Total number of connections (STARTUP events) per user type.

Figure 2.5: Classification of GridFTP connections (STARTUP events).



(a) Heatmap of the total number of connections (STARTUP events).



(b) Total number of Upload/Download Connections.

Figure 2.6: GridFTP connection by server locality and transfer direction (STARTUP events).

2.6 Use Case 1: Traffic Classification using SNAG

The need for accurate traffic classification at HCC arises from the necessity for monitoring/managing both encrypted and unencrypted traffic flows. GridFTP transfers use an encrypted control channel during the connection setup stage and use multiple randomly selected TCP channels for data movement. Due to the encrypted nature of the GridFTP control channel, traditional techniques like deep packet inspection (DPI) units *cannot* be employed to classify traffic. Apart from GridFTP transfers, other encrypted activity at HCC includes ssh-logins. Without

accurate and reliable traffic flow classification, *an undue burden is placed on the packet processing middleboxes* such as DPI units, firewalls and intrusion detection systems (IDS) due to the large amounts of data transferred. To alleviate this problem, we propose an application-aware SDN approach to traffic classification using SNAG. With the GridFTP application servers communicating application-layer metadata to the network-layer SDN controller in a secure and seamless fashion, accurate and reliable GridFTP traffic classification is possible. Further, accurate flow classification information can be employed to provide service differentiation and to actively manage the large-scale GridFTP flows from other network traffic as discussed in Section 2.8.

2.6.1 Deployment

At HCC, experiments such as CMS or LIGO rely heavily on high-throughput computing and both projects utilize GridFTP for WAN file transfers. In order to accurately classify GridFTP transfers from both the CMS and the LIGO projects, we deploy a production SDN environment on the project testbed as shown in Figure 1. Our architecture accurately classifies traffic from both CMS and LIGO projects. The traffic classification system was deployed to work with *thirteen (13) GridFTP servers*. Through SNAG, real-time traffic classification information of both CMS and LIGO projects is available to both the SDN controller and the monitoring system. Our approach not only provides fine-grained traffic classification information of project users, but also provides useful application-layer metadata through a secure RESTful API.

Our system classifies traffic by project association and user roles. Traffic flows from six different user types are identified by SNAG namely:

1. *CMS PhEDEx* – Represents the production CMS user data transfers initiated by the PhEDEx [48] data placement system. These transfers are associated with the movement of large physics datasets including ".root" files between sites.

2. *USCMSPool* – Represents data transfers associated with users' jobs, typically data used during an individual researcher's workflow. These transfers include both experimental data and the corresponding output logs.

3. *CMSProd* – These transfers are similar to *CMS PhEDEx*, however, it also include project-level information of CMS production workflows. Thus, these transfers represent a specific project's workflow rather than that of an individual researcher.

4. *LCG Admin* – Represent transfers associated with SAM (Site Availability Monitoring). They are small transfers designed to test the connectivity between different sites.

5. *LIGO* – These transfers represent LIGO transfers. LIGO transfer are generally opportunistic in nature and they share the HCC's networking resources at UNL.

6. *Other Users* – Apart from the user roles/types listed above, admin and test transfers are also initiated locally by site administrators and HCC staff.

User types $(1-4)$ represent CMS user roles, and user type (6) is local site transfers (both CMS and LIGO) by site administrators and HCC staff.

2.6.2 Results and Discussion

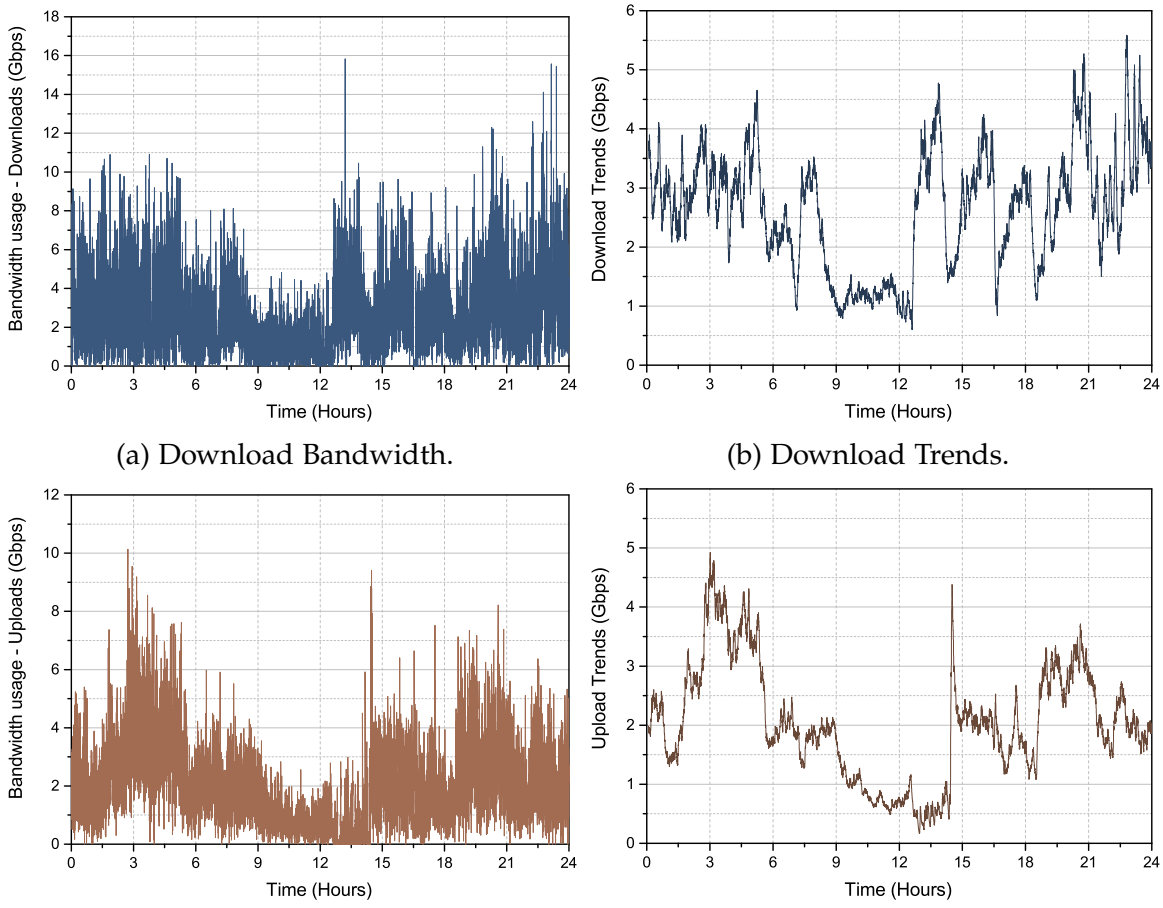
Our SNAG architecture classifies traffic not only by user/workflow information, but also based on the connection event type. Three types of connection events are defined namely: a) *STARTUP* events representing new GridFTP connections, b) *UPDATE* events to ensure that connections already established are kept alive, and c) *SHUTDOWN* events representing connection terminations. Figure 2.5a presents the total number of connection STARTUP events measured over a 6-month period classified by user type. A total of *171.69 million* connections were classified during the measurement period of 6 months. This is based on the *nonuple* (i.e. 9-tuple) information described in Section 2.4.1. Figure 2.6a shows the heatmap of the GridFTP connection information classified based on the GridFTP server (*S1* to *S13*) fulfilling the request, measured over a 6-month period. We note that all connections are load-balanced using a Linux Virtual Server (LVS) [49] load balancing system and is therefore independent of the server pool infrastructure capabilities. The per-server connection heatmap clearly shows that the load balancing system works as intended for the GridFTP server connection pool. The heatmap provides valuable information to the site operator about the instantaneous loads on each server in the server pool. Novel load-balancing techniques can be developed to cater to specific user/workflows types by exploiting the application-awareness information obtained from our SNAG architecture. Figure 2.5b shows the GridFTP connection distribution (only STARTUP events) classified by user type, and measured over a 6-month period. We see that *USCMSPool* users generate the bulk of the connections from analysis transfers of end-user jobs. At HCC, we see that approximately 90% of the connections established on a daily basis are generated by *USCMSPool* and *CMSProd* users.

Accurate and reliable traffic classification at the site-level provides valuable insights to the site operators about network resource usage and accounting. Our SNAG solution provides a fine-grained approach to traffic classification that is not possible using traditional approaches. By relying on real-time application metadata to ascertain flow information, we can develop novel and intelligent traffic classification techniques that incorporate a wider spectrum of classification parameters. Network administrators can use the traffic classification information from SNAG to make resource allocation decisions and in optimizing network services to improve transfer/security performance for end-users. This application-aware traffic classification technique is also vital to developing new monitoring views (Use Case 2), and in providing intelligence for active network management (Use Case 3).

2.7 Use Case 2: Network Monitoring using SNAG

Distributed, large-scale transfers from experimental science workflows such as CMS and LIGO at HCC rely on stable, secure and robust network infrastructure for normal operation. At HCC, we have identified the need for a real-time network monitoring system to understand end-user resource requirements, and to ensure smooth operation through effective and efficient network resource utilization. The SNAG architecture demonstrates an application-aware approach where applications collaborate with the network-layer to create monitoring views that *cannot* be realized using traditional network management approaches. SNAG generates unique monitoring views from application-layer metadata which can be used to understand user behavior. For example, a monitoring view created using application-awareness can help us understand how an opportunistic user, such as

LIGO, utilizes the shared networking resources. As distributed high-throughput computing workflows become data-intensive and flow continuously between sites, a careful accounting of resource usage is necessary for resource owners to be comfortable with opportunistic sharing. Monitoring views based on classified traffic as discussed in Section 2.6 can help us understand per-user or per-workflow resource usage including bandwidth consumption, network activity, connection trends etc. In the following sections, we present monitoring views, trend analysis and forecasting/estimation of GridFTP transfers by building on the traffic classification system presented in the previous section.



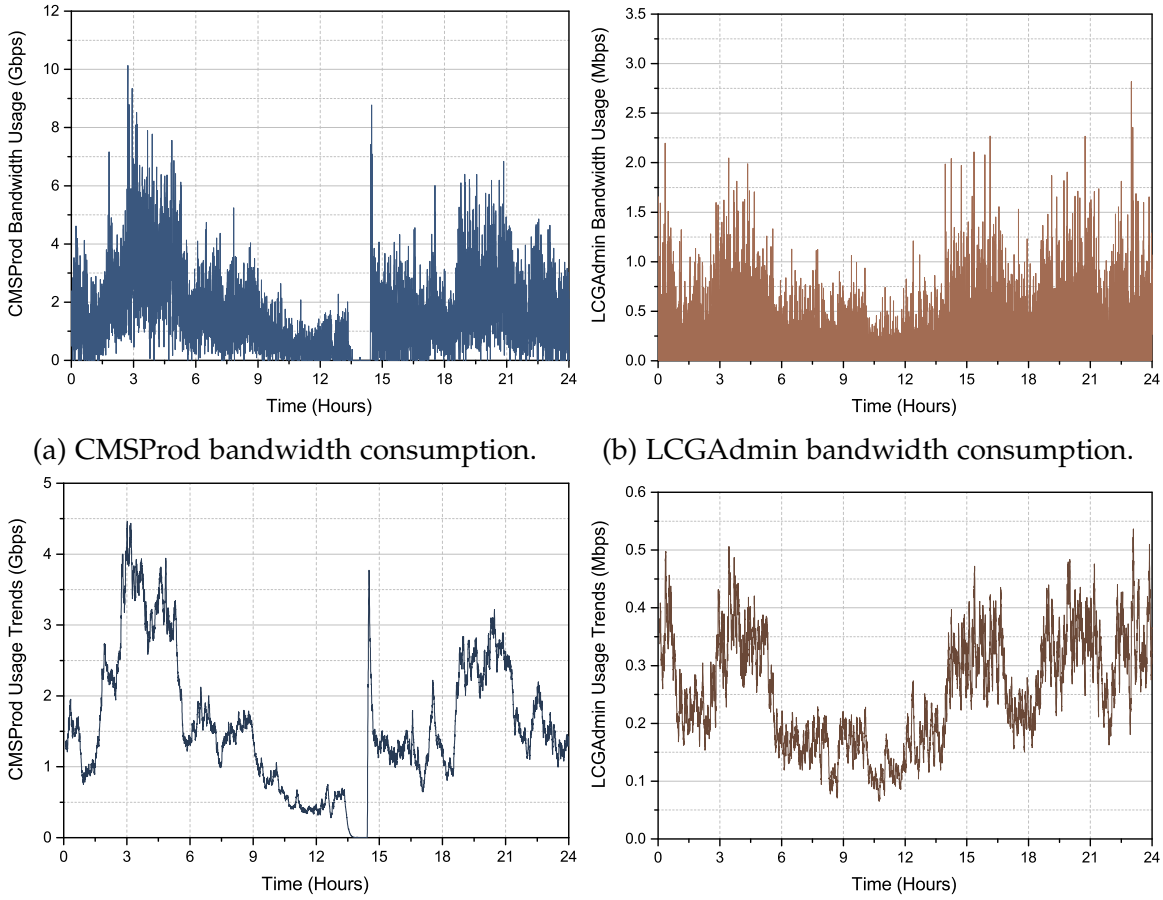
(a) Download Bandwidth.

(b) Download Trends.

(c) Upload Bandwidth.

(d) Upload Trends.

Figure 2.7: Bandwidth Utilization and Trends.



(a) CMSProd bandwidth consumption.

(b) LCGAdmin bandwidth consumption.

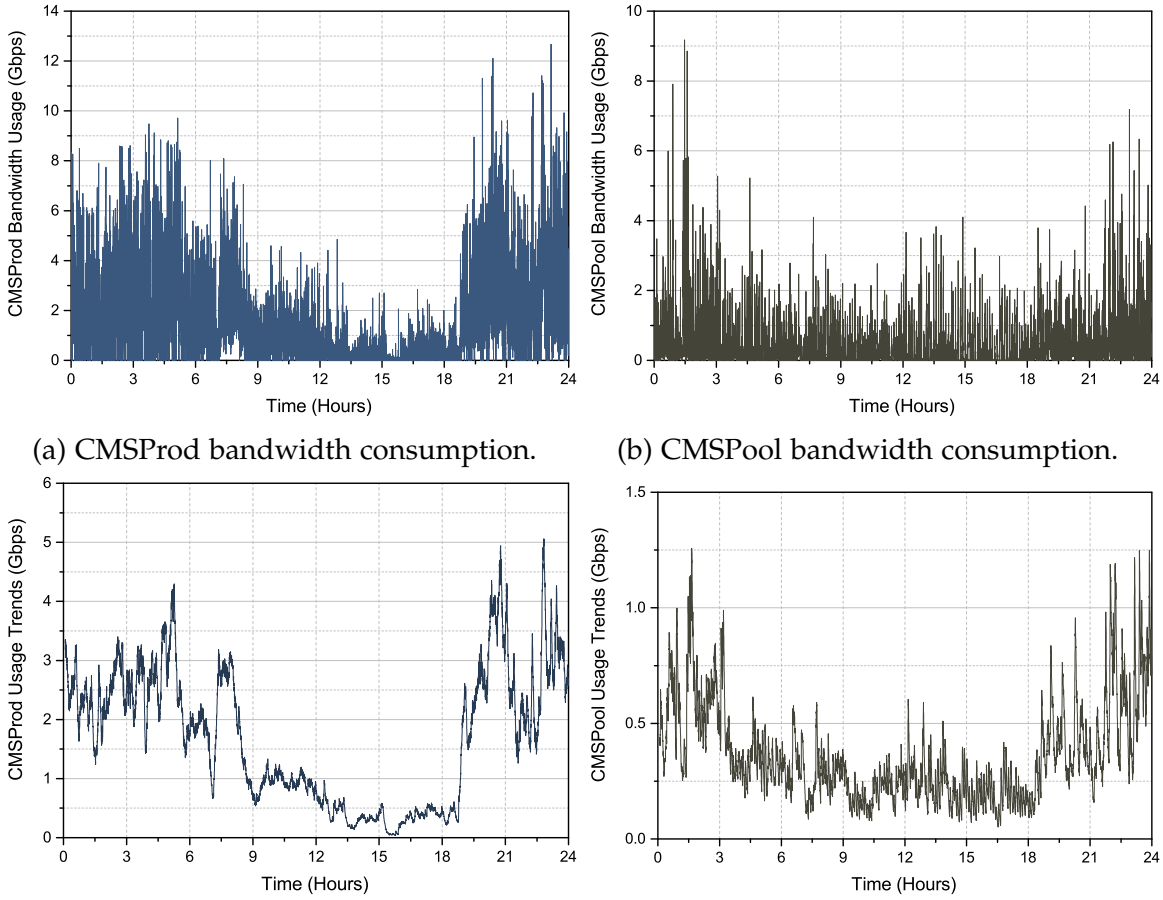
(c) Consumption trends - CMSProd.

(d) Consumption trends - LCGAdmin.

Figure 2.8: Upload bandwidth consumption by user type measured over 24 hours.

2.7.1 Monitoring Views and Trend Analysis

The SNAG architecture makes it possible to monitor the network in real-time, thus providing site-level monitoring views. Using the application-awareness information provided by SNAG, we can monitor in real-time, the number of upload/download connections initiated, per-user or per-experiment connection information, bandwidth consumption by different users/workflows by direction (upload/download), connection trends classified by user/workflow type and forecasting views for end-user connections. We employ simple moving average (SMA) to analyze the trends in bandwidth consumption and the trends in the



(a) CMSProd bandwidth consumption.

(b) CMSPool bandwidth consumption.

(c) Consumption trends - CMSProd.

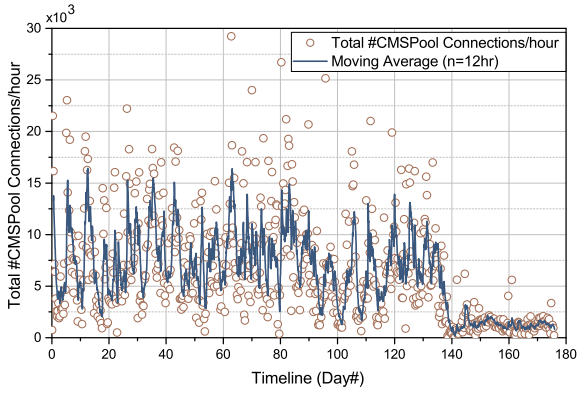
(d) Consumption trends - CMSPool.

Figure 2.9: Download bandwidth consumption by user type measured over 24 hours.

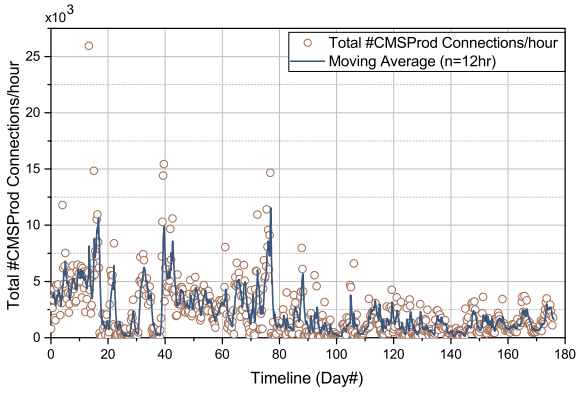
number of connections initiated by user/workflow type. We define the simple moving average $\bar{m}_{SMA}$ as follows:

$$\bar{m}_{SMA} = \frac{\bar{x}_M + \bar{x}_{M-1} + \cdots + \bar{x}_{M-(n-1)}}{n} = \frac{1}{n}\sum_{i=0}^{n-1} \bar{x}_{M-i} \tag{2.1}$$

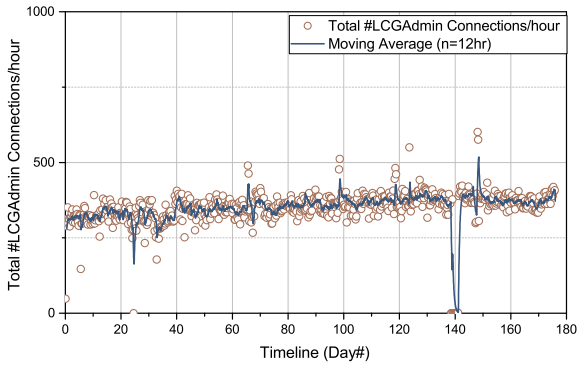
where, n is the sample size and M is the position of the current data point. We use the simple moving average to analyze bandwidth and connection trends presented in Section 2.7.3.



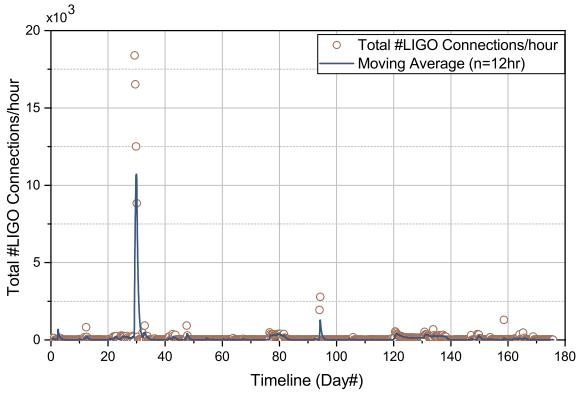
(a) Trends - CMSPool.



(b) Trends - CMSProd.

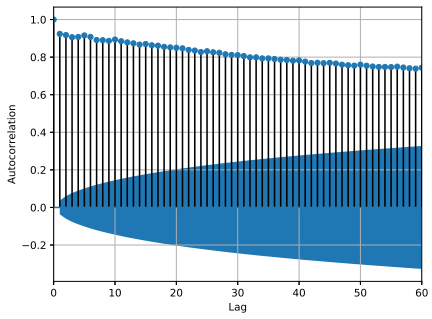


(c) Trends - LCGAdmin.

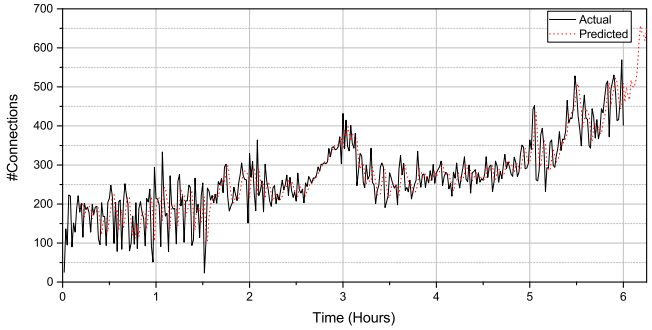


(d) Trends - LIGO.

Figure 2.10: Connection trends by user type.



(a) Correlogram showing the Autocorrelation properties of the traffic data.



(b) Forecasting GridFTP connections using Holt-Winters Model.

Figure 2.11: Prediction and Forecasting GridFTP connection STARTUPs.

2.7.2 Forecasting and Prediction

Adaptive models based on exponential smoothing can be applied in forecasting and prediction of data points in a time series that is optionally characterized by trends, seasonality, and periodic/random fluctuations. Time series data components can be combined in an additive, multiplicative or a mixed fashion as noted in [50]. The Holt-Winters model [51, 52] (or triple exponential smoothing) can be applied to forecasting time series data with seasonal components. Single exponential smoothing (equation 2.2), also referred to as Brown's model [53], can be used to obtain a single prediction data point $\hat{x}_t$. The term $\alpha \in (0, 1)$ represents the *smoothing coefficient* and its choice dictates the weight assigned to the most recent observed data point (i.e. x_t).

$$\hat{x}_t = \alpha \cdot x_t + (1 - \alpha) \cdot \hat{x}_{t-1} \tag{2.2}$$

Therefore, α represent a memory decay rate, with higher smoothing coefficients weighing the most recent observation heavily and placing little emphasis on historical events. We define the level ℓ to represent a single prediction, and the trend b_t is the slope of the series for two adjacent data points. Therefore, $b_t = x_t - x_{t-1}$. Double exponential smoothing is then equation 2.2 applied to both ℓ and b_t and is given by:

$$\ell_t = \alpha x_t + (1 - \alpha)(\ell_{t-1} + b_{t-1}) \tag{2.3}$$

$$b_t = \beta(\ell_t - \ell_{t-1}) + (1 - \beta) b_{t-1} \tag{2.4}$$

$$\hat{x}_{t+1} = \ell_t + b_t \tag{2.5}$$

The term $\beta \in (0,1)$ is the *trend coefficient*. By applying exponential smoothing to the seasonal components in addition to ℓ and b_t as discussed above, $t+p$ prediction data points can be obtained. The Holt-Winters model in additive form is as described below:

$$\ell_t = \alpha(x_t - s_{t-L}) + (1-\alpha)(\ell_{t-1} + b_{t-1}) \tag{2.6}$$

$$b_t = \beta(\ell_t - \ell_{t-1}) + (1-\beta)b_{t-1} \tag{2.7}$$

$$s_t = \gamma(x_t - \ell_t) + (1-\gamma)s_{t-L} \tag{2.8}$$

$$\hat{x}_{t+p} = \ell_t + pb_t + s_{t-L+1+(p-1)modL} \tag{2.9}$$

The term γ is the *seasonal change smoothing coefficient*. Further, we note that the initial value of the initial trend b_0 is obtained as:

$$b_0 = \frac{1}{L}\left(\frac{x_{L+1} - x_1}{L} + \frac{x_{L+2} - x_2}{L} + ... + \frac{x_{L+L} - x_L}{L}\right) \tag{2.10}$$

We use the Holt-Winters model to forecast traffic behavior and to estimate short-term [54] resource requirements.

2.7.3 Results and Discussion

Figure 2.6b shows the total number of upload/download connections for all connection event types (i.e. STARTUP, UPDATE and SHUTDOWN). The results presented in this section represents monitoring data measured over a 24 hour period. We further note that each data point represents an aggregate measured over a 10 second interval. However, it is also possible to measure the number of connections in real-time. Next, we present the total upload and download band-

width consumption (in Gbps) of all GridFTP users at HCC. This is an accurate and real-time bandwidth measurement computed using the file transfer information provided by the SNAG application-aware module and is representative of the total bandwidth utilization of all GridFTP transfers at HCC. Figures 2.7a and 2.7c show the total download and upload bandwidth consumption, and the Figures 2.7b and 2.7d show the trends in download and upload bandwidth consumption computed using a 15 minute simple moving average (SMA).

Total upload and download bandwidth consumption and their associated trends classified by user/workflow type is shown in Figures 2.8 and 2.9 respectively. Figures 2.8a and 2.8b presents upload bandwidth consumption for *CMSProd* and *LCGAdmin* users respectively. The corresponding users' bandwidth consumption trends are shown in Figures 2.8c and 2.8d respectively. Similar results for download bandwidth consumption are shown in Figure 2.9. We note that *LCGAdmin* user transfers represent small test transfer uploads and with the exception of *CMSProd* and *USCMSPool* users' transfers, no other transfers were initiated during the measurement period. In Figure 2.10, we present the connection trends classified by user/workflow type. Trends in the total number of connections established for individual users are presented in Figures 2.10a through 2.10d respectively. Lastly, in Figure 2.11, we present GridFTP connection forecasting and prediction using the Holt-Winters model presented in Section 2.7.2. The correlogram presented in the Figure 2.11a shows the autocorrelation properties of the GridFTP connection time series with a lag of one hour, i.e. $lag = 60$ (minutes). The correlogram shows strong positive autocorrelation properties. The shaded cone represents the 95% confidence intervals, and also corroborate the high autocorrelation for the time series. Figure 2.11b shows the forecast data over a 6 hour measurement period. We set the smoothing, trend, and seasonal change smoothing coefficients

to $\alpha = 0.7, \beta = 0.1$ and $\gamma = 0.9$ respectively. We see that the prediction closely models the actual measurement, and therefore, the Holt-Winters model can be employed to obtain reasonable GridFTP connection forecasting information.

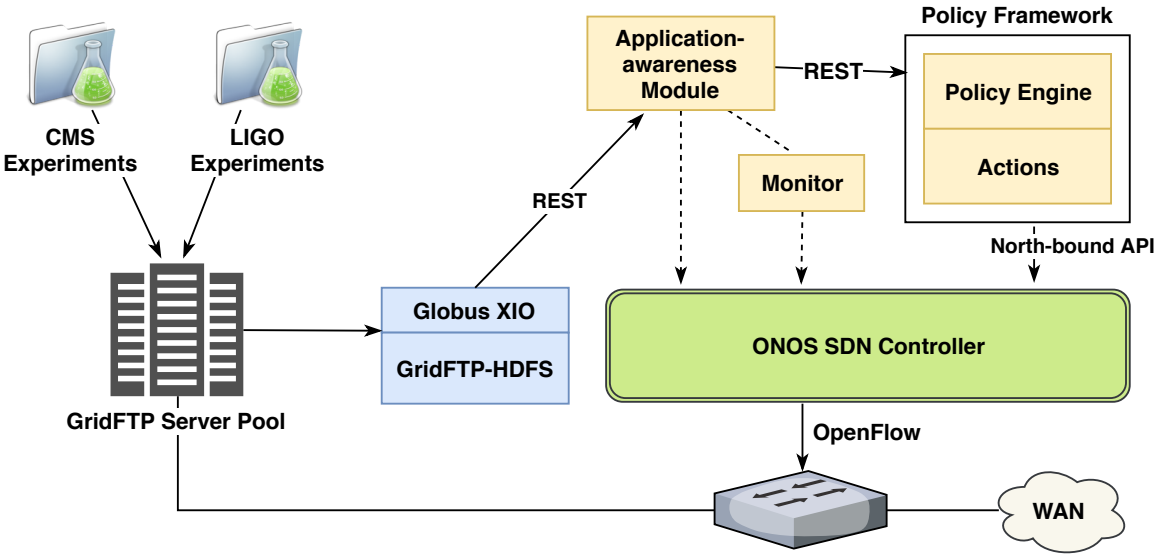


Figure 2.12: Solution Architecture.

Thus, real-time network monitoring of GridFTP users is possible through SNAG. By providing real-time monitoring and trends information to the site-operator, we can influence resource reservation and planning decisions. Further, automated alerts can be set up to notify the network administrators on the occurrence of specific events such as LIGO users initiating a large number of connections. We also demonstrated the usefulness of application-aware information in resource forecasting and prediction. We note that more sophisticated models for forecasting and modeling time series data can be employed with SNAG to accurately forecast end-user requirements. Accurate and reliable forecasting data can help network operators set bounds on networking, compute and storage requirements.

2.8 Use Case 3: Differentiated Network Services and Active Network Management using SNAG

Network services differentiation [55] is essential at the site-level to enhance transfer performance at the network edge. With both CMS and LIGO projects using the same network infrastructure, the inability to classify and differentiate services results in low-priority users of one workflow blocking/preempting the high priority users of the other. Without service differentiation, the network operator cannot optimize users' transfers within a workflow. Although each connection can reserve resources, our solution instead relies on developing site-level improvements to optimize data transfer performance. Our goal is to facilitate the network operator to account for site-wide resources accurately. To further motivate the problem, we provide two crucial drivers for the need for application-aware services differentiation.

2.8.0.1 Application-aware traffic prioritization

GridFTP connections rely on the use of encrypted control channels and parallel TCP streams (randomly chosen) for data movement. Application-awareness allows an operator to manage both GridFTP and non-GridFTP flows. Traffic classification using Layer-2 or Layer-3 approaches alone cannot serve to differentiate encrypted traffic between the same endpoints. Consider two transfers between the same endpoints; a low-priority user can easily preempt a high-priority user by consuming significant resources opportunistically. Application-awareness is a crucial driver for such cases as application metadata can be easily exploited by an operator to prioritize the appropriate data transfer.

2.8.0.2 Policy-driven service differentiation

Network operators routinely face problems with resource usage accounting for data-intensive science workflows like CMS and LIGO. This difficulty arises from an inability to monitor each experiments' traffic and map it to resource usage. In the campus network, multiple stakeholders are responsible for the operation and maintenance of its network infrastructure. Therefore, a policy-driven mechanism for managing and monitoring resources proves vital. Furthermore, such policies can serve to regulate/meter resource usage and ensure the prioritization of scheduled transfers over opportunistic transfers.

2.8.1 Differentiated Services Solution Approach

Application-awareness combined with SDN forms the basis of our solution for policy-driven management of data-intensive science workflows. First, we begin with a reliable and accurate classification of flows from different experiments. This objective is accomplished through the exchange of application metadata between the GridFTP application servers and the SDN controller. The SDN controller creates and manages a repository of all ongoing transfers obtained from the GridFTP servers. This information is used to make network-layer and data-plane decisions. A policy framework is used by the network operator to define and apply the appropriate policies to these transfers. Our framework consists of a policy-engine and an associated policy specification language implemented as an SDN application. The SDN controller uses the policy controls and translates them to appropriate data plane decisions. Based on the defined policy, a set of actions (that correspond to the policy strategy) are applied to the data flows.

2.8.2 Policy Framework

Our policy framework contains a policy engine, a specification language and a set of defined actions. The policy engine, implemented as an SDN application, is responsible for managing user-defined policies. Our policy specification language uses JavaScript Object Notation (JSON). The policy engine processes a set of actions to affect flow treatment to appropriate transfers. Management of policies is enabled using RESTful APIs. In the following, we briefly discuss each of these components:

2.8.2.1 Policy Engine

The policy engine comprises of a *policy manager*, an *event handler*, a *parser*, and a *policy repository*. The *policy manager* is responsible for policy lifecycle management (including those created using the RESTful APIs). After the *parser* validates each policy, it is stored in the *repository* (both new and updates). The policy engine's *event handler* converts the policy specifications to corresponding OpenFlow rules that are programmed to the switches using the south-bound APIs.

2.8.2.2 Specification Language

The policy specification uses JSON resources encapsulated in a REST POST to interact with the policy engine in the framework. The specification language allows us to define/modify the default flow treatment for an experiment. Each policy specifies an action type and to which data transfers it is applicable. The policy framework can apply actions to user transfers, workflows or experiments. Further, each policy has a lifecycle defined by start- and end-times for policy enforcement. Next, we describe actions and associated service differentiation strategies for data transfers.

2.8.2.3 Actions and Strategies

Actions along with predefined strategies are responsible for implementing the desired flow treatment behaviors in the data-plane. A site operator can define different QOS and traffic prioritization/management strategies, and specify the appropriate strategy to be applied using the policy specification. Each strategy is composed of two parts: *definitions* (e.g. queues to use, associated priorities, max/min rate settings etc.) and a *trigger* that enforces the policy.

2.8.3 Differentiated Services Solution Architecture using SNAG

The solution architecture is as shown in Figure 2.12. The GridFTP server pool oversees data transfers from both CMS and LIGO projects. The GridFTP server uses the XIO callout module to send application metadata to the SNAG application on the SDN controller. SNAG is responsible for providing information regarding new and ongoing data transfers to the site operator. The site operator then uses the transfer statistics to make the policy decisions that are enforced using the policy framework built as an SDN application. The policy framework provides a RESTful API for communication, and policy enforcement can either be performed manually by the site operator or can be automated by the SDN controller.

2.8.4 Algorithm Design for Service Differentiation

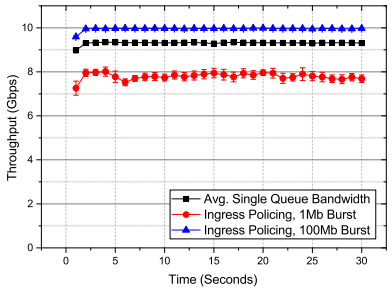
Our algorithm focuses on providing a policy-based solution to network services differentiation. We rely on two important principles namely: *application-awareness* and *policy strategy*.

While application-awareness gives us valuable insights into the current state of data transfers from various users/projects/experiments, policy strategy, on

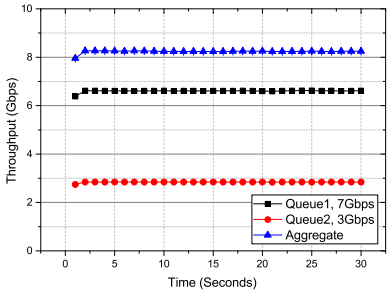
Algorithm 2.1 AA-DNS(p, *target*)

Require: Policy File (p), *target*
Output: Provisioned Network Flows
Initialization :
for all $p \in P$ **do**
  Generate $p\_id(p)$
  if ($p \notin PolicyMap$) **then**
    Add $(p, p\_id(p))$ to $PolicyMap$
  else
    Replace $PolicyMap(p)$
  end if
end for
Policy Enforcement :
for all $p \in PolicyMap$ **do**
  if ($p.expired \neq$ `TRUE`) **then**
    `CONFIGURE`($queues \in p.strategy$)
    $flowrule =$ `COMPOSE`$(p, target)$
    $Ruleset \leftarrow flowrule$
    for all Switch $s \in S$ **do**
      `APPLY` $Ruleset(s)$
    end for
  end if
end for

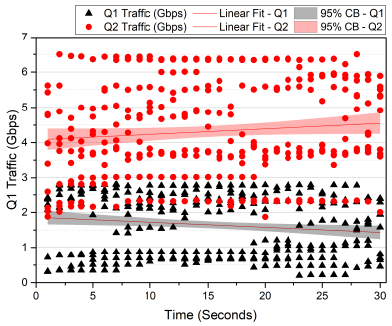
the other hand, allows us to apply the right forwarding behaviors to the desired flows. The *AA_DNS(p, target)* algorithm is as shown in Algorithm 1. The algorithm initializes policy IDs for each policy and manages them in a *PolicyMap*. The policy engine parses each policy and extracts the specified strategy (*p.strategy*). The policy strategy contains information about QoS requirements, queue specifications and priority definitions for the *target* traffic. This information is used to create the appropriate flow rules by the SDN controller and applied to the corresponding switches in the data plane to change the forwarding behaviors.



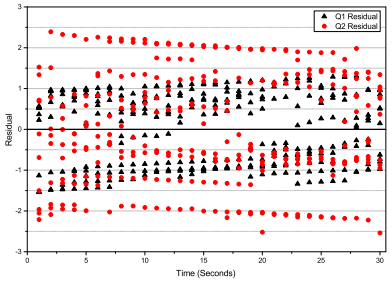
(a) 10 Gbps Bottleneck Bandwidths.



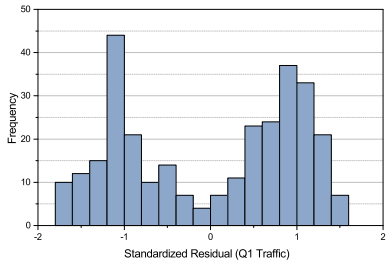
(b) Two Queues with Traffic Shaping, Ratio: 7-3.



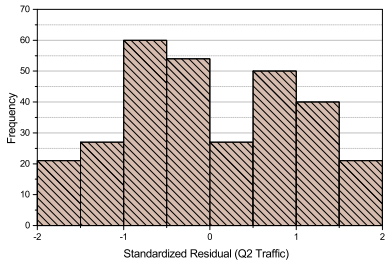
(c) RIQ performance, Random Traffic, 10 Samples.



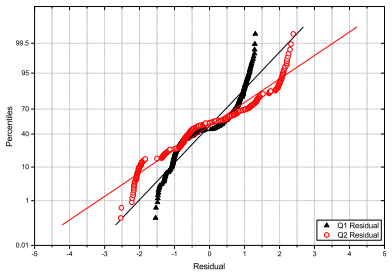
(d) Standardized residuals (Q1 and Q2 traffic).



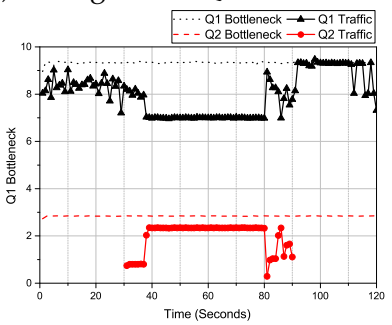
(e) Histogram of Q1 distribution.



(f) Histogram of Q2 distribution.



(g) Normal probability plot of residuals.



(h) RSQ switching performance with LIGO Traffic.

Figure 2.13: Differentiated network services queuing performance.

2.8.5 Implementation

Our SNAG differentiated services solution is used to manage and monitor data transfers over the GridFTP protocol, providing differentiated network services. The policy framework and the QoS/prioritization systems are implemented as an ONOS application. SNAG provides application metadata that is not only used to obtain real-time data transfer information but also allows for the creation of a GridFTP transfer statistics repository. This repository provides useful information that aids the site operator in designing/choosing the appropriate policy for service differentiation.

We use the hierarchical token bucket (HTB) [56] for egress traffic shaping. HTB is a egress queuing discipline implementation for Linux kernel packet scheduler user space utilities. We limit our discussion to egress traffic shaping (using queues) and do not use ingress rate limiting algorithms (that employ policing) to provide differentiated services. Ingress rate limiting/policing does not use queues but drops packets beyond a certain rate instead; this is problematic as some protocols react severely to dropped packets. We present two strategies for implementing differentiated network services for GridFTP transfers using egress traffic sharing and queues.

2.8.5.1 Resource Isolated Queues (RIQ)

This strategy creates separate resource isolated queues for each project type based on the SNAG classification information. We evaluate the transfer performance on a 10 Gbps link. For example, we create two queues $q1$ and $q2$, with ingress traffic shaped to 7 Gbps and 3 Gbps respectively (i.e. a 7:3 ratio). The two queues have equal priority and have their priorities set to a value higher than the best-effort

link. We evaluate this strategy by placing CMS traffic on $q1$ and LIGO traffic on $q2$. We note that the queue capacity ratios can be adapted to the resource requirements of the individual projects.

2.8.5.2 Resource Switched Queues (RSQ)

In this strategy, the traffic from a predefined project is automatically switched to a sub-queue on arrival. The sub-queue limits the resources utilized by a given project. For example, we create two queues, $q1$ and $q2$ with rates of 10 Gbps (main queue) and 3 Gbps (sub-queue) respectively. The priority of $q1$ is greater than that of $q2$, which is an important difference from the previous strategy. In this approach, we utilize a single high priority 10 Gbps queue for all data transfers except LIGO. LIGO transfers if initiated, are *automatically switched* to use $q2$. In the following section, we discuss the performance of both strategies.

2.8.6 Results and Discussion

In Figure 2.13a, we show the bottleneck bandwidths for creating the 10 Gbps queue when both egress traffic shaping and ingress rate limiting are used. It can be seen that the performance of the ingress rate limiting varies with the burst size. Thus, we focus only on egress traffic shaping for our evaluations. Figure 2.13b shows both the individual and aggregate bottleneck bandwidths for resource isolated queuing (RIQ) strategy with a 7:3 ratio (i.e. $q1$=7 Gbps and $q2$=3 Gbps). The performance of RIQ for 10 random transfers is shown in Figure 2.13c. It can be seen that both queues exhibit linear correlation with a decrease in $q2$ traffic resulting in a corresponding increase in $q1$ traffic as shown by the trend lines (plotted with 95% confidence intervals). The standardized residual plot for RIQ is shown in Figure 2.13d. The histogram plots of the residuals for both $q1$ and $q2$

are also shown in the Figures 2.13e and 2.13f respectively. These results show that the transfer datasets are symmetric and bi-modal with no outliers. The bi-modal nature of the transfer dataset is also established from the normal probability plot of residuals as shown in Figure 2.13g. Since both queues are set to use the same priorities, they behave as two rate-limited best-effort queues that are independent of each other. Figure 2.13h shows the performance of RSQ, where LIGO traffic is switched to a lower priority queue on arrival. It can be seen that during the LIGO transfers, the higher priority queue $q1$ shapes its traffic to accommodate LIGO flows. Note that $q2$ uses only a portion (max 3 Gbps) of the larger 10 Gbps queue i.e. $q2$ does not represent a separate queue from $q1$. Comparing the aggregate throughputs of both strategies from Figures 2.13b and 2.13h, we see that strategy 2 provides an additional capacity improvement of 11.74%. This is because CMS flows can achieve higher throughputs as fewer traffic shaping requirements free more bandwidth.

Thus, SNAG can be used successfully to make intelligent decisions for active network management. The use of application-layer metadata provides greater flexibility in making network-layer decisions that are adaptive to end-user requirements. Using application-awareness, we can provide differentiated network services, network isolation, and improved resource utilization for network-layer flows.

2.9 Recommendations for Building Application-aware Architectures

Our application-aware architecture can be easily extended to obtain application-layer metadata from other applications. Example application servers include web

servers, file servers, storage servers and (high-performance) file-system layers, cloud/grid management servers, and media streaming services among others. Our exemplary architecture facilitates the exchange of application-layer metadata with the network-layer securely using RESTful APIs over the HTTPS protocol. However, our proposed approach is not limited to the use of RESTful APIs for application- and network-layer communication. We can facilitate the exchange application-layer metadata using other platform-independent mechanisms such as remote procedure calls (e.g., gRPC), NETCONF/YANG, message queue protocols (e.g., RabbitMQ, SOAP, etc.), and SDN controller-specific north-bound APIs.

Application-aware architectures are beneficial for both active and passive network management. An operator can easily integrate application-aware decision-making into network traffic classification, monitoring, trend analysis, and forecasting tasks. By employing a high-performance data store (e.g., an Elastic stack cluster), advanced data analytics, real-time alerts, alarms, and auditing/reporting is possible. Integration with big-data analytics, business intelligence, and enterprise reporting tools can provide critical inputs for policy/decision makers.

2.10 Conclusions and Future Work

We proposed SNAG, an application-aware SDN solution for at-scale GridFTP traffic classification, monitoring, and management. SNAG demonstrates a cross-layer (application- and network-layer) collaborative approach to create traffic classification and monitoring views that are not achievable using traditional layering approaches. We also demonstrate how to exploit application-layer metadata to make intelligent network management decisions. At HCC, this is crucial in helping us understand how opportunistic users from LIGO, utilize the shared network

resources. As distributed high-throughput computing workflows become data-intensive and flow continuously between sites, a careful accounting of resource usage is necessary to ensure that resource owners are comfortable with opportunistic sharing. Although SNAG has focused on GridFTP integration as it constitutes the majority of the transfers at HCC, our approach is specific to GridFTP. HCC performs an increasing amount of transfers through the XRootD and HTTP protocols. The implementation also called "XROOTD" [10], is pluggable and can be integrated with SNAG.

We also present an application-aware SDN approach to network services differentiation and active network management. Using this approach, we can apply per-workflow policies at the site-level which was not previously possible. We demonstrated an application-driven mechanism for creating workflow-specific resource queues. Using application-aware network management principles, site operators can optimize resource allocation and fine-tune network performance to suit end-user application requirements. Further, our approach can benefit other network management tasks including routing/forwarding, traffic analysis/redirection, resource provisioning, and QoS.

Our future work will focus on creating adaptive strategies on-the-fly so that we can automate policy enforcement for network management decisions. There are paths for the SNAG project to grow: we would like to apply the SNAG application-awareness approach to other applications and add a more significant percentage of the transfer servers to the SNAG architecture. Transferring the first petabyte of data through SNAG-managed network infrastructure will be an important internal milestone.

Chapter 3

Optimized Service Chain Mapping and Reduced Flow Processing with Application-Awareness

Network Function Virtualization (NFV) brings a new set of challenges when deploying virtualized services on commercial-off-the-shelf (COTS) hardware. Network functions can be dynamically managed to provide the necessary services on-demand and further, services can be chained together to form a larger composite. In this chapter, we address an important technical problem of mapping service function chains (SFCs) across different data centers with the objective of reducing the flow processing costs. The SFC mapping problem is critical to enhancing the virtualized service networks' performance, as it places high demands on the performance of these service functions. We develop an integer linear programming (ILP) formulation to optimally map service function chains to multiple data centers while adhering to the data center's capacity constraints. We propose a novel application-aware flow reduction (AAFR) algorithm to simplify the SFC-ILP to significantly reduce the number of flows processed by the SFCs. We perform a thorough study of the SFC mapping problem for multiple data centers and evaluate the performance of our proposed approach with respect to three parameters: i) impact of number of SFCs and SFC length on flow processing cost, ii) capac-

itated/uncapacitated flow processing cost gains, and iii) balancing flow-to-SFC mappings across data centers. Our evaluations show that our proposed AAFR algorithm reduces flow-processing costs by 70% for the capacitated-SFC mapping case over the SFC-ILP. In addition, our uncapacitated AAFR (AAFR-U) algorithm provides a further 4.1% cost-gain over its capacitated counterpart (AAFR-C).

3.1 Introduction

Recently, several evolutionary trends in networking such as software defined networking (SDN) and network functions virtualization (NFV) have had a significant impact on the next-generation of network architectures. SDN introduces the network control and data plane separation and allows dynamic and programmatic control of the network via open interfaces. NFV, on the other hand, utilizes traditional server virtualization techniques to provide an architecture where network functions run over commercial-off-the-shelf (COTS) hardware rather than dedicated boxes.

In traditional networks, a set of network middleboxes (e.g. firewalls, NATs, IDSs, etc.) process traffic flows, with each flow traversing the middleboxes in a specific order. Service function chaining (SFC) represents a way of stitching together diverse network/service functions (i.e. middleboxes) to form a composite service. In order to implement SFC, network operators have to steer the traffic flows to a set of hardware middleboxes in the local network. This process is expensive to deploy and maintain [57]. SFC with virtualized network/service functions provides greater flexibility in deploying services by outsourcing the middlebox functionality to the cloud. Processing traffic on virtualized service functions and SFCs place additional demands on the underlying cloud and network

infrastructure. Traffic steering through the virtual infrastructure must guarantee performance and quality of service. Since virtualization is an added layer of abstraction, flows can incur additional delays. Unlike traditional middleboxes that are optimized to perform specific tasks, virtualized services rely on commodity hardware for flow processing. Additional latencies are introduced by outsourcing network/service functions to the cloud as traffic flows will have to traverse the WAN for processing. This is a serious concern for applications that are delay-constrained and rely on fast/transparent processing by network functions during data transfers.

Despite NFV presenting many new opportunities, challenges exist in NFV placement and SFC mapping across data centers. Placing network/service functions far from the user results in additional delays. Security services provided via NFV infrastructure for large networks are expected to support context-aware and low-latency applications in a highly efficient manner. The introduction of security services at the network edge, while reducing the response time, may impact core-network utilization. To address the above challenges, we study the SFC mapping and placement problem across multiple data centers in this work. We focus on solving the SFC mapping problem to provide security services to both interactive and large-volume data transfers. We model our solution to suit an existing 100G production network topology, which is an important difference compared to previous works. We also ensure that the model is general enough to be applicable to other multi-data center network scenarios.

The main contributions of this chapter are the following: i) We formulate an integer linear programming (ILP) problem (SFC-ILP) for optimized SFC mapping in a multi-data center topology and propose an application-aware flow reduction (AAFR) algorithm to reduce the NFV flow processing workloads. Traffic and

resource characteristics of a testbed network consisting of four data centers are employed in the proposed ILP model, ii) We compare the results of AAFR with the SFC-ILP formulations for a production U.S. CMS Tier-2 network, iii) We conduct extensive performance evaluations of the proposed AAFR algorithm and show flow processing workload savings of $47\% - 70\%$ can be achieved for the capacitated-SFC mapping problem, and iv) We present a quantitative comparison of the capacitated- and uncapacitated-SFC mapping problems and show that SFCs with unlimited processing capacities do not result in significant cost benefits. Finally, we demonstrate the AAFR algorithm's effectiveness in balancing flow-to-VNF (virtual network function) mappings to avoid SFC-loading problems.

The rest of this chapter is organized as follows: Section 3.2 provides an overview of our virtualized services model and describes the network scenario targeted by our solution; Section 3.3 presents an ILP formulation of the VNF placement problem; Section 3.4 extends the VNF placement problem to formulate the SFC mapping optimization problem; Section 3.5 gives an overview of application-awareness and presents our proposed application-aware flow reduction (AAFR) algorithm; Section 3.6 describes our experimental setup, testbed data center network setup, and performance results; some related works are presented in Section 3.7; lastly in Section 3.8, we conclude our work.

3.2 Virtualized Services Model and Network Scenario

3.2.1 Virtualized Services Model

Network functions in general and security services in particular are modeled as virtual network functions (VNFs). We denote the set of VNFs by V. Each VNF $v \in V$ is hosted on a physical node $n \in N$ and consumes a fraction of the node's

capacity C_n. Further, a VNF v has a constrained traffic processing capacity as defined by C_v. Multiple VNFs are combined together (i.e. "chained") in a defined order to form a service function chain (SFC) $s \in S$. Each SFC is mapped to a node $n \in N$ in the physical network substrate. Thus, all ingress traffic flows $f \in F$ are processed by a set of SFCs S in the virtual network substrate to provide network services to these flows.

3.2.2 Network Scenario

The physical network substrate is composed of two data centers, connected to a common border router. We model the physical network substrate as a directed graph $G^P(N, E)$ composed of a set of physical nodes and links. While the nodes host the VNFs that are then networked to form a service function chain, the virtual network traffic is carried over the links in the physical network substrate. The physical nodes and links are hosted on commercial off-the-shelf (COTS) hardware and have processing and transfer capacity constraints. The physical network substrate forms the NFV infrastructure (NFVI). The virtual network substrate is also modeled as a directed graph $G^V(V, E_v)$ and consists of VNFs and their associated links.

The network scenario used by our optimization model is as shown in Figure 3.1. The proposed work models the service function chain mapping to data centers across multiple campus networks. The SFCs are set up only in the data centers 1 and 2. Each data center generates two types of traffic: i) Traffic from experimental science projects such as CMS [7] and LIGO [24], and ii) Commodity Internet traffic from users in the campus network. Data centers DC1 and DC2 are managed by the same network and connect to the wide area network (WAN) through the Brocade MLXe border router as shown in Figure 3.1. This combined network hosts

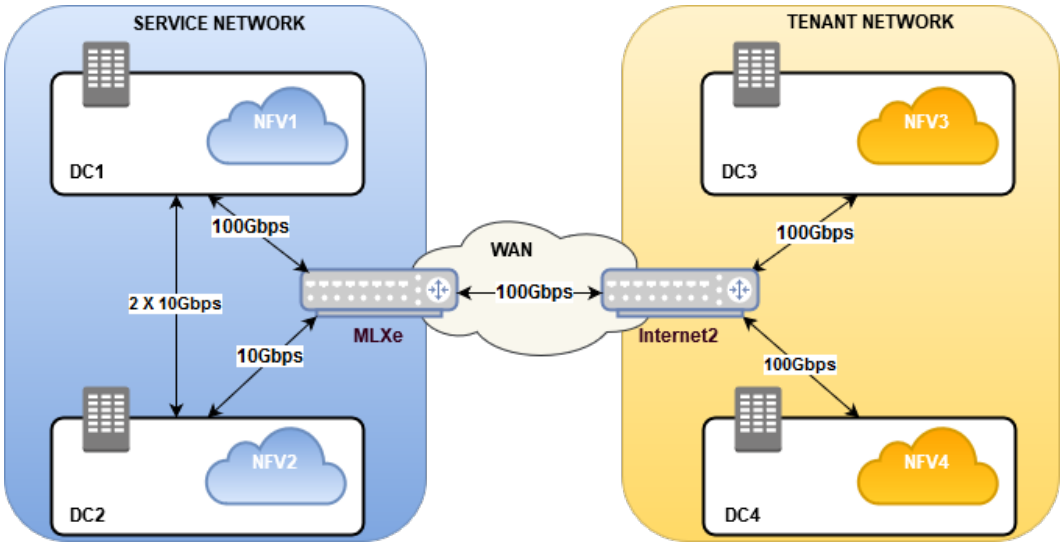


Figure 3.1: Network Scenario.

the NFV infrastructure (NFVI) and provides the service functions for processing both experimental science and commodity Internet traffic. Two other data centers connect to this network over the Internet2 backbone.

3.3 VNF Placement Problem

In this section, we present a formulation of the VNF placement problem. This formulation forms the basis for our SFC mapping problem discussed in Section 3.4. The VNF placement problem is defined as follows:

Definition 3.1: *Given a physical network substrate graph $G^P(N, E)$, find the optimal placement of VNFs for maximizing admission of flows $f \in F$.*

We present a mathematical model for the VNF placement problem below. The model determines the optimal VNF placement for maximizing the number of flows admitted into the virtual network for processing. Each flow f is associated with forwarding policy pol_f, and is admitted for VNF processing only if a *flow-to-VNF*

mapping is possible. Depending on where a VNF is placed in the flow's path as dictated by a forwarding policy, resource usage, network utilization and flow processing latencies will be impacted. Thus, VNF placement is an important problem and presents many challenges including VNF deployment in geographically separated data centers, flow routing consistency to satisfy forwarding policy requirements and efficient *flow-to-VNF* mapping to minimize resource usage.

3.3.1 Problem Formulation: VNF-LP

3.3.1.1 Decision Variables

We define the following binary decision variables for the VNF placement problem.

$$\alpha_f = \begin{cases} 1, \text{ if flow } f \text{ is admitted for processing.} \\ 0, \text{ otherwise.} \end{cases} \tag{3.1}$$

$$\beta_{v,n}^{u} = \begin{cases} 1, \text{ if VNF instance } u \text{ of } v \text{ placed on a} \\ \quad \text{physical node } n \in N. \\ 0, \text{ otherwise.} \end{cases} \tag{3.2}$$

$$\gamma_{e}^{e_q,f} = \begin{cases} 1, \text{ if flow } f \text{ to } e_q \in E_q \text{ is routed through} \\ \quad \text{a physical link } e \in E. \\ 0, \text{ otherwise.} \end{cases} \tag{3.3}$$

$$\delta_{v,n}^{u,f} = \begin{cases} 1, \text{ if flow } f \text{ is assigned to instance } u \text{ of } v \\ \quad \text{placed on a physical node } n \in N. \\ 0, \text{ otherwise.} \end{cases} \tag{3.4}$$

Table 3.1: Notations used in the LP Formulation.

Parameters	Description
$G^P(N, E)$	Graph representing the physical network substrate.
$G^V(V, E_v)$	Graph representing the virtual network substrate.
$G^S(V^s, E_v^s)$	The SFC graph with nodes V^s on the chain and E_v^s virtual links between them.
N	The set of nodes in the physical network substrate.
E	The set of links in the physical network substrate.
V	The set of VNFs. Each VNF $v \in V$ denotes a particular type of VNF (e.g. DPI, Firewall etc.).
E_v	The set of links in the virtual network substrate.
V^s	The set of VNFs on the SFC.
E_v^s	The set of virtual links on the SFC.
V_{end}^s	The set of endpoint nodes (ingress and egress) of the SFC i.e. $\{v(in), v(out)\}$.
V_{fn}^s	The set of VNFs on the SFC excluding V_{end}^s.
F	Denotes the set of ingress traffic flows, with each flow $f \in F$.
S	Denotes the set of service chains.
C_n	The resource capacity of the physical node n.
C_v	Processing capacity a VNF v.
$B_{i,j}$	The bandwidth of the physical link $(i, j) \in E$.
B_v	The bandwidth capacity of a virtual link $e_v \in E_v$.
B_s	The bandwidth capacity of the SFC $s \in S$.
K_u	The maximum allowed VNF instances u of type $v \in V$.
K_v	The maximum allowed VNFs on the network.
K_s	The maximum allowed SFCs on the network.
b_f	Bandwidth demand of the flow $f \in F$.
pol_f	Represents the forwarding policy that is applied by the SFC to flow $f \in F$.
$T_{i,j}$	Propagation latency of the physical link $(i, j) \in E$.
T_v	Processing latency due to VNF $v \in V$.
T_s	Processing latency of the SFC.
T_s^{max}	Maximum tolerable latency of the application/user due to processing by the service chain $s \in S$.

3.3.1.2 Objective

To maximize the number of flows admitted by the network for VNF processing.

$$\text{Maximize} \sum_{f \in F} \alpha_f \tag{3.5}$$

The objective in (3.5) is subject to the constraints described in the following.

3.3.1.3 Constraints for VNF Placement

$$\sum_{n \in N} \beta_{v,n}^{u} \leq 1, \qquad \forall 1 \leq u \leq K_u, v \in V \tag{3.6}$$

$$\sum_{u=1}^{K_u} \sum_{n=1}^{N} \beta_{v,n}^{u} \leq K_u, \qquad \forall v \in V \tag{3.7}$$

$$\sum_{v \in V} \sum_{u=1}^{K_u} \beta_{v,n}^{u} \cdot C_v \leq C_n, \qquad \forall n \in N \tag{3.8}$$

Constraint (3.6) allows a maximum of one active instance of a particular VNF type per physical node. Constraint (3.7) ensures that all instances u of a particular type of VNF v does not exceed the allowed maximum total number of instances for that type. Constraint (3.8) states that the demand requirements of all VNF types should not exceed the processing capacity of the physical node.

3.3.1.4 Constraints for flow-to-VNF mapping

$$\sum_{n \in N} \sum_{u=1}^{K_u} \delta_{v,n}^{u,f} \leq 1, \forall f \in F, u \in pol_f \tag{3.9}$$

$$\delta_{v,n}^{u,f} \leq \beta_{v,n}^{u}, \forall f \in F, \forall n \in N, \forall 1 \leq u \leq K_u, u \in pol_f \tag{3.10}$$

$$\sum_{n \in N} \sum_{f \in F} \delta_{v,n}^{u,f} \cdot b_f \leq C_v, \qquad \forall 1 \leq u \leq K_u, v \in V \tag{3.11}$$

$$\beta_{v,n}^{u} \leq \sum_{f \in F} \delta_{v,n}^{u,f}, \quad \forall n \in N, v \in V, 1 \leq u \leq K_u \tag{3.12}$$

Constraint (3.9) ensures that we have a maximum of one flow per VNF instance. Constraint (3.10) ensures that the number of flows should not exceed the total number of active VNF instances. Constraint (3.11) guarantees that the flows processed do not exceed the maximum processing capacity of the VNF. Constraint (3.12) makes sure that the VNFs are not instantiated if there are no flows in the network. Thus, the number of VNFs instantiated in the network do not exceed the total number of flows.

Table 3.2: Binary Decision Variables.

Binary Decision Variables for VNF Placement Problem	
α_f	Binary variable for flow admission.
$\beta_{v,n}^{u}$	Binary variable for VNF placement.
$\gamma_{e}^{e_q,f}$	Binary variable for routing a flow f to a virtual link $e_q \in E_q$ through physical substrate link $e \in E$.
$\delta_{v,n}^{u,f}$	Binary variable for flow assignment to a VNF.
Binary Decision Variables for SFC Mapping Problem	
$\beta_{v,n}$	Binary variable for VNF placement.
$\beta_{s,n}^{v'}$	Binary variable for VNF mapping to SFC on a physical node.
$\lambda_{s,e}^{e_v}$	Binary variable for mapping a virtual link to a physical link.
$\delta_{s,n}^{f}$	Binary variable for mapping a flow to a SFC.
$\mu_{s,n}^{e_1,e_2}$	Binary variable for checking edges converging to a node.

3.3.1.5 Flow conservation constraints

$$\sum_{i,j \in E} \gamma_{i,j}^{e_q,f} - \sum_{i,j \in E} \gamma_{j,i}^{e_q,f} = \sum_{u=1}^{K-u} \delta_{v(in),n}^{u,f} - \sum_{u=1}^{K-u} \delta_{v(out),n}^{u,f}, \quad \forall f \in F, e_q \in E_q, i, j \in N \tag{3.13}$$

$$\sum_{i,j \in E} \gamma_{i,j}^{e_q,f} - \sum_{i,j \in E} \gamma_{j,i}^{e_q,f} = \begin{cases} \alpha_f, & \text{if } i = v(in) \\ -\alpha_f, & \text{if } i = v(out) \\ 0, & \text{otherwise} \end{cases} \tag{3.14}$$

$$\sum_{f \in F} \gamma_{i,j}^{f} \cdot b_f \leq B_{i,j}, \quad \forall (i,j) \in E \tag{3.15}$$

Constraints (3.13,3.14) represent flow conservation constraints. Also, constraint (3.15) constrains the flow capacity of the network to not exceed the network's link capacity.

The VNF placement problem is similar to the well-known class of NP-hard problems: *facility location* and *generalized assignment*. In the following section, we extend the VNF placement problem and formulate the SFC mapping problem.

3.4 SFC Mapping Problem

A network operator's strategy for service provisioning across data centers should provide a convenient mechanism for i) the placement and deployment of virtual network functions, ii) decisions regarding the mapping of service function chains to VNF instances and iii) how the ingress traffic flows are routed to the appropriate service chains. In this section, we present a model for mapping SFCs for optimizing their placement cost across multiple data centers. The model can be used to make *flow-to-SFC* mapping decisions, while minimizing the deployment and placement cost for the network operators.

3.4.1 Assumptions

We make the following assumptions with our optimization model formulation:

- A complete (i.e. end-to-end) service corresponds to one SFC modeled as a single line graph, comprising of ingress and egress nodes, service nodes (VNFs), and consecutive concatenated links.
- To simplifiy the model formulations, we assume that the flow requests are decoupled from the VNFs.
- The endpoints of all the SFCs are known *a priori* and have fixed location corresponding to a node on the physical network substrate.
- The service function forwarders (SFFs) and their associated flow classifiers (FCs) are assumed to be known *a priori*.
- SFCs serve aggregated traffic of a set of users requesting a specific service from a specific location on the physical network substrate.
- Single aggregate (e.g. CPU, memory, etc.) resource type is assumed.
- SFC endpoints are assumed to be virtual nodes that are mapped to the physical nodes, although endpoints can be physical nodes.

3.4.2 Service Function Chaining Model

In this section, we present a formulation of the SFC mapping problem. It is defined as follows:

Definition 3.1: *Given a physical network substrate graph $G^P(N, E)$, find the optimal placement of VNFs on a service function chain to minimize the placement cost.*

We denote by S the set of all service function chains (SFCs) in the network. Each service chain s comprises of a set ($V' \subseteq V$) of VNFs “chained together" in some predefined order through virtual links. The SFC is also connected to

a set of ingress/egress endpoints (mapped to physical nodes) that are responsible for forwarding traffic through the chain. The service function path (SFP) is formed by the set of VNFs encompassing the chain. Service classifiers are defined at the ingress node and are tasked with mapping incoming flows to appropriate SFPs. Each service chain request is characterized by a flow request, a data rate requirement and a maximum tolerable delay specification. The SFC is therefore a simple line graph $G^S(V^s, E_v^s)$, where $V^s = \{v(in), v(s) \subset V, v(out)\}$ and $E_v^s = \big((v(in), v_1), (v_1, v_2), \ldots, (v_n, v(out))\big)$, with $E_v^s \subseteq E_v$. To differentiate between endpoint nodes and function nodes on the service chain, we denote the set of endpoint nodes as $V_{end}^s = \{v(in), v(out)\}$, and the set of function nodes as $V_{fn}^s = \{v'\} = V^s - V_{end}^s$.

3.4.3 Network Model

The physical network substrate and the virtual network substrate are modeled as before since there is a high correlation between the VNF placement problem and the SFC mapping problem.

3.4.4 Problem Formulation: SFC-LP

3.4.4.1 Decision Variables

We define the following binary decision variables for the SFC mapping problem.

$$\beta_{v,n} = \begin{cases} 1, \text{ if VNF } v \text{ is placed on a physical node} \\ \quad n \in N. \\ 0, \text{ otherwise.} \end{cases} \tag{3.16}$$

$$\beta_{s,n}^{v'} = \begin{cases} 1, \text{ if VNF } v' \text{ mapped to SFC } s \text{ is placed} \\ \quad \text{on a physical node } n \in N. \\ 0, \text{ otherwise.} \end{cases} \tag{3.17}$$

$$\lambda_{s,e}^{e_v} = \begin{cases} 1, \text{ if virtual link } e_v \text{ on } s \text{ is mapped to a} \\ \quad \text{physical link } e \in E. \\ 0, \text{ otherwise.} \end{cases} \tag{3.18}$$

$$\delta_{s,n}^{f} = \begin{cases} 1, \text{ if flow } f \text{ is mapped to service chain} \\ \quad s \text{ on node } n \in N. \\ 0, \text{ otherwise.} \end{cases} \tag{3.19}$$

$$\mu_{s,n}^{e_1,e_2} = \begin{cases} 1, \text{ if edges } e_1 \text{ and } e_2 \text{ of service chain } s \\ \quad \text{converge on node } n \in N. \\ 0, \text{ otherwise.} \end{cases} \tag{3.20}$$

3.4.4.2 Objective

To minimize the cost of SFC placement.

$$\text{Minimize} \sum_{n \in N} \sum_{v \in V} \beta_{v,n} + \sum_{e \in E} \sum_{s \in S} \sum_{e_v \in E_v^s} \lambda_{s,e}^{e_v} \tag{3.21}$$

The objective in (3.21) is subject to the constraints below.

3.4.4.3 Constraints for SFC placement

$$\beta_{s,n}^{v'} \leq \beta_{v,n}, \quad \forall n \in N, s \in S, v \in V, v' \in V^s \tag{3.22}$$

$$\sum_{n \in N} \beta_{s,n}^{v'} \leq 1, \quad \forall s \in S, v' \in V^s \tag{3.23}$$

$$\sum_{n \in N} \beta_{v,n} \leq K_v, \quad \forall v \in V \tag{3.24}$$

Constraint (3.22) ensures that the VNF must be available before we can place it on the SFC. Constraint (3.23) makes sure that only one VNF per SFC is mapped to one physical node. (3.24) ensures that all VNF instances in the network do not exceed a defined maximum.

$$\begin{aligned} \sum_{n \in N} \beta_{s,n}^{v(in)} \leq 1, \text{ and } \sum_{n \in N} \beta_{s,n}^{v(out)} \leq 1, \\ \forall s \in S, v(in), v(out) \in V_{end}^s \end{aligned} \tag{3.25}$$

Further, constraint (3.25) ensures that we only have one ingress and one egress endpoint per service chain.

3.4.4.4 Constraints for Resource Capacity

$$\sum_{v \in V} \beta_{v,n} \cdot C_v \leq C_n, \quad \forall n \in N \tag{3.26}$$

$$\sum_{v \in V} \beta_{s,n}^{v'} \cdot B_s \leq \beta_{v,n} \cdot B_v, \quad \forall n \in N . \forall v' \in V^s \tag{3.27}$$

$$\sum_{s \in S} \sum_{e_v^s \in E_v^s} \lambda_{s,e}^{e_v} \cdot B_s \leq B_{i,j}, \quad \forall e \in E \tag{3.28}$$

The resources requested by the VNFs cannot exceed the resource capacity of the physical node that they are mapped to, as presented in constraint (3.26). Further, in constraint (3.27), we ensure that the VNFs have sufficient traffic processing

capabilities to handle the traffic from all SFC that they are mapped to, and finally, constraint (3.28) ensures that the bandwidth capacity of the physical link is sufficient to handle the traffic from all SFCs.

3.4.4.5 Constraints for Flow-to-SFC mapping

$$\sum_{n \in N} \sum_{s=1}^{K_s} \delta_{s,n}^{f} \leq 1, \qquad \forall f \in F \tag{3.29}$$

$$\beta_{s,n}^{v'} \leq \sum_{f \in F} \delta_{s,n}^{f}, \quad \forall n \in N, v' \in V^s, 1 \leq s \leq K_s \tag{3.30}$$

$$\sum_{n \in N} \sum_{f \in F} \delta_{s,n}^{f} \cdot b_f \leq B_v, \qquad \forall 1 \leq u \leq K_u, v \in V \tag{3.31}$$

In constraint (3.29), we ensure that there is only one flow mapped to an SFC. In (3.30), the number of flows that are mapped to SFCs are constrained to not exceed the number of service chains. The bandwidth capacity of service chain is lower-bounded by B_v as in constraint (3.31).

3.4.4.6 Constraints for Flow Conservation

Constraints (3.32,3.33) represent the flow conservation constraints for the SFCs. This is similar the flow conservation constraints (3.13,3.14) for the VNF placement problem.

$$\sum_{e \in E} \lambda_{s,e_1}^{e_v} - \sum_{e \in E} \lambda_{s,e_2}^{e_v} = \sum_{f \in F} \delta_{s,n}^{v(in),f} - \sum_{f \in F} \delta_{s,n}^{v(out),f}, \quad \text{where } e_1 = (i,j) \text{ and } e_2 = (j,i) \tag{3.32}$$

$$\sum_{e \in E} \lambda_{s,e_1}^{e_v} - \sum_{e \in E} \lambda_{s,e_2}^{e_v} = \begin{cases} \alpha_f, & \text{if } e_1 = v(in) \\ -\alpha_f, & \text{if } e_2 = v(out) \\ 0, & \text{otherwise} \end{cases} \tag{3.33}$$

$$\sum_{e_1,e_2 \in E_v^s} \mu_{s,n}^{e_1,e_2} \leq 1, \forall n \in N, s \in S; \quad \text{and } \forall e_1, e_2 \in E_v^s \mid e_2 \in E_v^s - \{e_2\} \tag{3.34}$$

Finally, the constraint (3.34) requires that each set of virtual links connected to a VNF on a service chain s can only be mapped to a single node $n \in N$ in the physical network substrate. It can be seen that the SFC mapping problem is an extension of the VNF placement problem.

Algorithm 3.1 The SFC-LP Mapping Algorithm

1: Solve the (SFC-LP) and find the mapping of $G^S(V^s, E_v^s)$ to $G^p(N, E)$ and do the following:
2: Compute the mappings of SFC $s \in S, \forall n \in N$ to satisfy resource/capacity constraints C_n and C_v.
3: Obtain the *flow-to-SFC* allocations $\forall f \in F$ and assign a subset of the flows $\{f_i\} \rightarrow s_i, \forall s_i \in S$.
4: Setup service function chains $\sum_{j=1}^{|S_i|} s_i, \forall s_i \in S, \forall i \in \{\text{DCs}\}$ with a total of $S_{i,j}$ SFCs in data center i.

The cost function in (3.35) is used to estimate the total processing cost of a single flow by a service function chain mapped to a data center as allocated by the SFC-LP algorithm.

$$\mathcal{C}_{\mathcal{SFC}-\mathcal{LP}} = \mathcal{C}_f + \mathcal{C}_v \tag{3.35}$$

where,

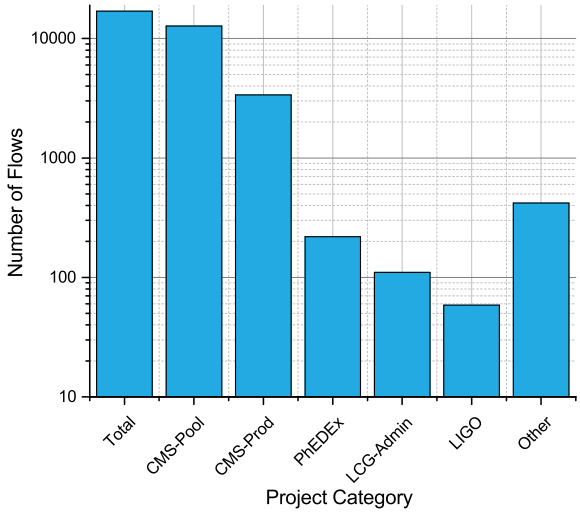
$$\mathcal{C}_v = \mathcal{C}_B \cdot \omega_B + \mathcal{C}_l \cdot \omega_l + \mathcal{C}_h \cdot \omega_h \tag{3.36}$$

The total cost estimate $\mathcal{C}_{\mathcal{SFC}-\mathcal{LP}}$ is a function of fixed and variable costs, with the fixed cost $\mathcal{C}_f$ depending on the number of VNFs in the SFC. The variable cost $\mathcal{C}_v$ depends on the flows' impact on bandwith, latency and associated host (i.e. $\mathcal{C}_B, \mathcal{C}_l, \mathcal{C}_h$) costs respectively. Each component of the variable cost $\mathcal{C}_v$ has an associated weight function (i.e. $\omega_B, \omega_l,$ and ω_h) to account for variations in link bandwidth, link latency and the host resource capacity of each data center. Figure 3.2 shows the various parameters used in the cost computation. These parameters were obtained from measurements across four data centers. Figure 3.2a shows the number of experimental science transfers at data center 1 averaged over a period of one month categorized by project type. Approximately 15000 experimental science flows are processed by data center 1 every day. The available bandwidth between data centers and the corresponding round-trip time (RTT) measurements are as shown in Figures 3.2c and 3.2b. The data center network setup is detailed in Section 3.6.1.

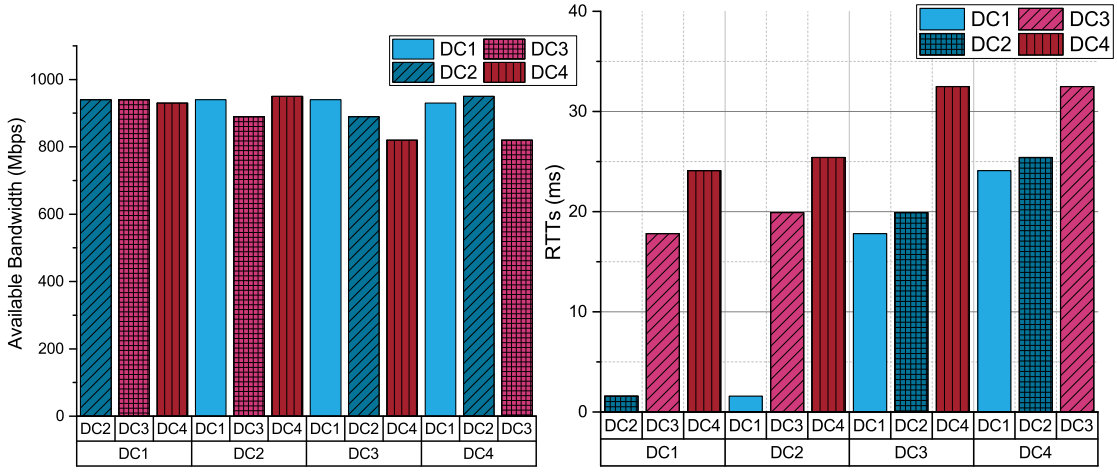
In our measurements for the data center links DC1-DC3, DC1-DC4, DC2-DC3 and DC2-DC4, we observed approximately 25%, 2% and 10% variations in available bandwidth, RTTs and associated host processing costs respectively. These observed variations were modeled as an uniform random distribution and incorporated in the corresponding weight functions.

3.5 Application-Aware Flow Reduction (AAFR)

Application-awareness is achieved using the SDN-managed Network Architecture for GridFTP transfers (SNAG) proposed in [16]. We define application-awareness as the exchange of application-layer metadata with the network-layer, thereby facilitating collaboration between the two layers. SNAG exposes an application



(a) Average number of Experimental Science Transfers per day.



(b) Available Bandwidth between data centers as measured by Iperf3.

(c) Average RTT between data centers.

Figure 3.2: Experimental setup parameters for different data centers.

program interface (API) and communicates the application-layer metadata associated with the underlying connections over a representational state transfer (REST) interface. The traffic classification information in Figure 3.2a cannot be obtained *without* application-awareness since GridFTP [9] protocol uses encrypted TCP sessions between end-points for data movement.

In our work, we use SNAG to accurately identify and to differentiate experi-

mental science transfers from the commodity Internet traffic. Thus, application-awareness results in reduced flow processing workload for the SFCs as pre-classified experimental science traffic are not subject to flow processing by the SFCs in the NFVI. Subjecting only commodity Internet traffic to SFC processing is justified since experimental science workflows incorporate multiple security components to establish user/service identity. These components (such as X.509 PKI and proxy certificates) are used to protect communication between end-points and determine user credentials and authorization for specific actions. Thus, application-awareness reduces the flow-processing workload by subjecting only commodity Internet traffic to SFC processing. Therefore, the flow-processing cost of the AAFR algorithm accounts only for flow-switching and associated host costs as no resources are allocated for processing end-to-end experimental science traffic. However, commodity Internet traffic flows incur the same cost as before. Thus, AAFR reduces the flow processing costs by mapping SFCs only to commodity Internet traffic.

The AAFR mapping algorithm is shown in Algorithm 3.2. The algorithm updates the variable cost parameter ($\mathcal{C}'_v$) as follows:

$$\mathcal{C}_{\mathcal{AAFR}} = \mathcal{C}_f + \mathcal{C}'_v \tag{3.37}$$

where,

$$\mathcal{C}'_v = \mathcal{C}_{sw} \cdot + \mathcal{C}_h \cdot \omega_h \tag{3.38}$$

Algorithm 3.2 The AAFR Mapping Algorithm

for all $\{f_i\} \subset F$ **do**
  Construct a flow set F_e
  if $SNAG(f_i ==$ TRUE) **then**
    $F_e \leftarrow F_e \cup \{f_i\}$
    $F' \leftarrow F - F_e$
  end if
end for
for all $\{f_i\} \subseteq F'$ **do**
  Solve the (SFC-ILP) on f_i and find the mapping of $G^S(V^s, E^s_v)$ to $G^p(N, E)$.
  Update $\mathcal{C}'_n$ and $\mathcal{C}'_v$ to reflect the new capacity requirements.
end for
Compute the mappings of SFC $s \in S, \forall n \in N$ to satisfy the updated resource/-capacity constraints $\mathcal{C}'_n$ and $\mathcal{C}'_v$.
Obtain the *flow-to-SFC* allocations $\forall f_i \in F'$ and assign a subset of the flows $\{f_i\} \rightarrow s_i, \forall s_i \in S$.
Setup service function chains $\sum_{j=1}^{|S_i|} s_i, \forall s_i \in S, \forall i \in \{\text{DCs}\}$ with a total of $S_{i,j}$ SFCs in data center i.

3.6 Experimental Study

In this section, we evaluate the performance of the SFC-ILP and the AAFR algorithms. We solve the optimization problems described in Sections 3.3 and 3.4 using IBM CPLEX 12.7.1. We create a testbed data center network and use its traffic and resource characteristics to provide inputs to the optimization model. We then use the SFC mapping solutions provided by the models for SFC placement and associated flow processing. The following section outlines the data center network setup.

3.6.1 Data Center and Network Setup

We set up four data centers as shown in Figure 3.1. Each data center is set up on a high-performance server hosting an OpenStack Ocata cloud and SFC extensions for NFV management. Data centers DC1, DC3 and DC4 run one compute and

two controller nodes, with 8 cores and 128GB RAM on each node. DC2 is hosted on a shared private cloud with 80 VCPUs and 448GB of RAM. DC1 and DC2 communicate with each other over the WAN through the same border gateway router, and connect to the other data centers over the Internet2 backbone.

3.6.2 Results and Discussion

We employ real-world flow measurements at the Holland Computing Center (HCC), University of Nebraska-Lincoln to model the traffic flow characteristics between data centers. The flow costs are generated randomly based on the cross-data center parameters presented in Figure 3.2. A total of 15000 flows with random costs serve as inputs to the optimization models. Of the 15000 flows processed per day on average, about $70\% - 80\%$ (randomly chosen) flows constitute experimental science transfers with the remaining $20\% - 30\%$ forming the commodity Internet traffic. Each node in the physical network substrate is assumed to have a fixed resource capacity aggregate that is shared equally by all of the SFCs set up on that data center. The total number of VNFs per SFC is assumed to be constant for all SFCs in the network across all data centers. Unless otherwise specified, the flow processing capacity is limited to the total number of flows (F) and shared equally between each SFC (i.e. F/n flows/SFC for a total of n SFCs) in the network. A fixed per-flow processing cost associated with the SFC setup on each data center is added to the cost computation on each run. The NFVI for hosting SFCs are created only on data centers DC1 and DC2 (*service networks*), with data centers DC3 and DC4 forming the *tenant networks*. Thus, flows that are both internal and external to the *service networks* are processed by the DC1 and DC2. In this section we present the performance results of the SFC-ILP and the AAFR mapping algorithms.

Figure 3.3 shows the flow processing cost performance of the two algorithms

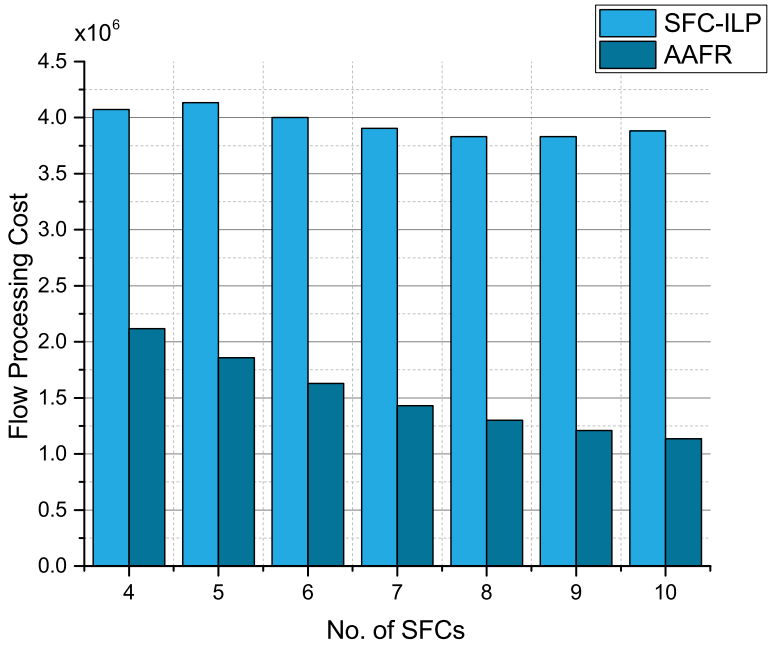


Figure 3.3: Cost Comparison.

for increasing number of SFCs in the *service network*. The number of SFCs in the *service network* is varied between 4 and 10 per data center and the flow processing cost is computed for both algorithms. We show that costs reduce with increasing number of SFCs in both cases. Increasing the number of SFCs in a data center reduces flow queuing and facilitates faster parallel processing leading to lowered costs. However, since SFC-ILP processes a larger number of flows it shows lower gains (about 4.9%) from increasing the number of SFCs compared to AAFR which shows gains of about 47%. In comparison to SFC-ILP, AAFR shows reduced cost gains between $47\% - 70\%$ for the same number of SFCs for both algorithms. We present the execution time performance for both algorithms in Figure 3.4. Both algorithms perform comparably, with AAFR performing about 5% faster than SFC-ILP.

We present the flow processing cost for the capacitated and uncapacitated versions of both algorithms in Figure 3.5. For the capacitated case, we limit the

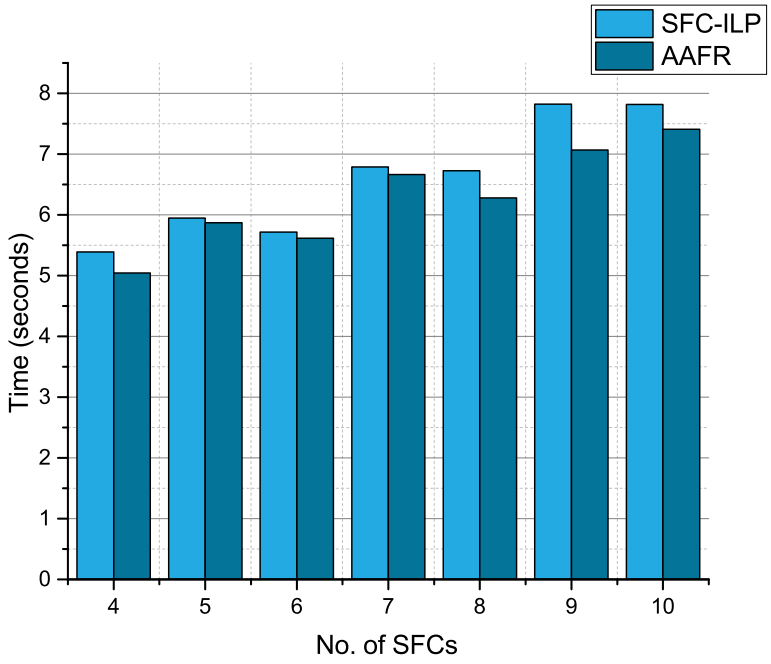


Figure 3.4: Performance Comparison.

flow processing capacity of each SFC equally to F/n for F flows and n SFCs in the network. This results in a SFC mapping problem that is similar to the capacitated facility location problem wherein each facility has a fixed processing capacity. We refer to this as the *capacitated-SFC* mapping problem (SFC-ILP-C and AAFR-C). We compare the above to an *uncapacitated-SFC* mapping problem (SFC-ILP-U and AAFR-U). Our evaluations show limited gains when each algorithm is compared to its uncapacitated counterpart i.e. SFC-ILP-C vs. SFC-ILP-U and AAFR-C vs. AAFR-U. The cost-gains are about 0.6% and 4.1% for the SFC-ILP and the AAFR algorithms respectively. For the AAFR algorithm, the 4.1% cost-gain is in addition to the $47\% - 70\%$ gains described before when per-SFC processing capacity limits are increased to large values. Thus, increasing the SFC flow processing capacity to an arbitrarily large value does not lower flow processing costs.

The impact of the number of VNFs per SFC on mapping costs are as shown in

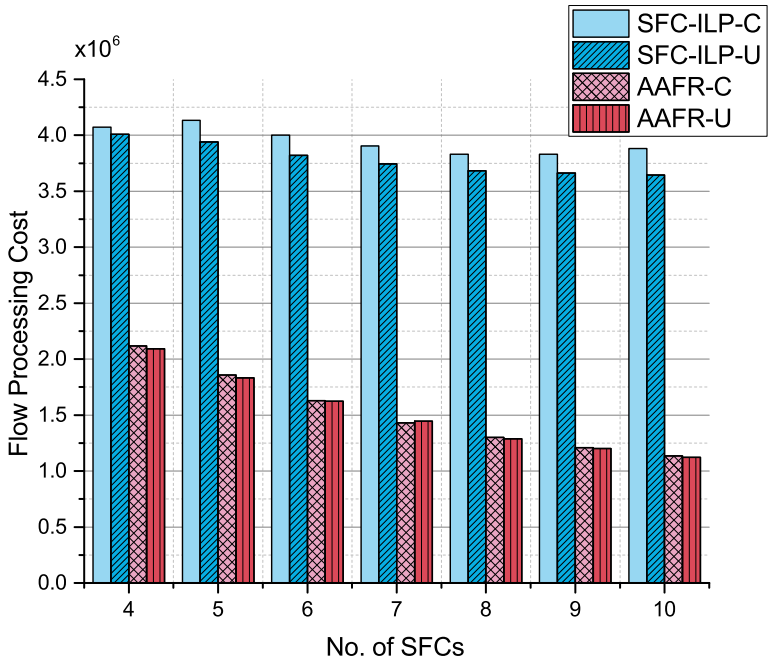


Figure 3.5: Variable and Fixed capacity costs.

Figure 3.6. The SFC mapping costs increases monotonically with a corresponding increase in the number of VNFs per SFC. The number of VNFs for each service chain is varied between 2 and 7 and we show that the costs increase by about 18.7% on average per additional VNF in the service chain for both algorithms.

In Figure 3.7, we present the flow distribution across SFCs in each data center for both algorithms. This evaluation is an extension of the *uncapacitated-SFC* mapping problem described above. The evaluation is presented for the $\#SFC = 4$ case for each data center in the *service network*. The result in Figure 3.7 shows the cumulative average number of flows placed in each SFC. We see that about 95.9% of the flows were mapped to the first three SFCs in the case of SFC-ILP, whereas only 70.25% of flows were mapped in the case of AAFR. Therefore, the majority of the flow processing load falls on roughly two-thirds of the SFCs in the network when SFC-ILP is used. Thus, AAFR is better at load balancing and mapping flows

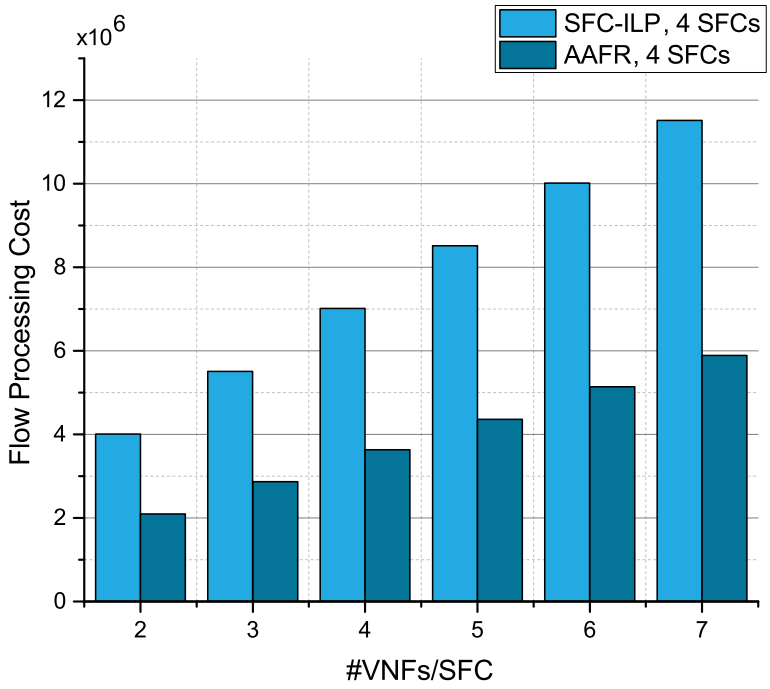


Figure 3.6: Impact of VNFs per SFC on costs.

to SFCs in each data center due to reduced flow-processing workloads compared to SFC-ILP. Thus, our evaluations show that the AAFR algorithm effectively reduces flow processing workloads across data centers by using application-aware flow classification.

3.7 Related Work

Recently, numerous works have focused on NFV resource allocation (NFV-RA) that deals with allocating network infrastructure resources demanded by virtual network services. A comprehensive survey is provided in [58]. VNF resource allocation looks at VNF forwarding graph embedding (VNF-FGE) and VNF scheduling (VNF-SCH) problems, with solutions focusing on embedding virtual networks into a physical network substrate. A few recent works [59, 60, 61], focus on VNF placement problems. The authors in [59] propose a near-optimal solution for the

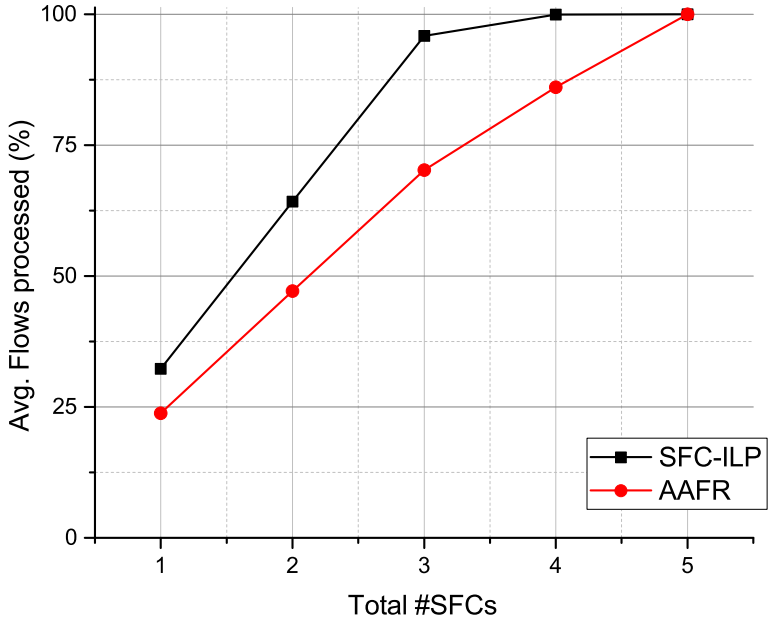


Figure 3.7: Cumulative average #flows processed by SFCs.

NFV location problem. They evaluate the solution performance with respect to setup and distance costs, and provide a bi-criteria solution within the specified capacity constraints. The authors in [60, 61] look at efficient VNF placement to aid service function chains. Both works propose ILP formulations for the VNF placement problem. The optimization models focus on VNF placement resilience and end-to-end latency reduction. Other works such as [62, 63] focus on the placement/resource allocation of specialized VNFs (e.g. vFirewalls or vDPIs). The authors in [62] model vDPI placement as a ILP multi-commodity flow problem and propose a greedy heuristic to reduce the placement cost. The work in [63] looks at optimizing resource allocation for elastic VNFs in cloud environments.

SFC placement/embedding problems are discussed in [64, 65, 66, 67, 68]. The work in [64] looks at how to optimize the deployment of SFCs for new users while balancing and readjusting the existing users' SFC to minimize deployment costs. A column generation scheme is designed based on service feasibility solution of

the ILP to approximate its solution. An ILP formulation and a heuristic algorithm for QoS guaranteed service function chain placement across multiple clouds is proposed in [65]. Each function of the SFC is modeled as a hidden markov model state transition and most-likely state sequence prediction is used to decide which VNFs have to be outsourced to a cloud. The proposed heuristic results in cost savings for VNF placement in the cloud compared to local placement. A service function selection algorithm is proposed in [66] to balance a service function's path distance and its load. The proposed algorithm considers load, latency and QoS class constraints to select service function paths during the initial deployment of the SFC across multiple data centers. Joint topology design and SFC mapping for Telco clouds is explored in [67] with the objective of minimizing bandwidth consumption. The proposed method uses feedback from critical sub-topology mappings to optimize SFC mapping. The work in [68] proposes an MILP solution for joint optimization of the different phases on NFV-RA i.e. NFV composition, VNF-FGE and VNF-SCH. An online NFV tools is developed using a heuristic-based algorithm.

Application-awareness with SDN is a fairly new area of research. Although deep packet inspection (DPI) can be employed to achieve application awareness, it is not applicable to encrypted transports. Therefore, we limit our application-awareness discussion to works that implement seamless application metadata exchange (without resorting to DPI) between application- and network-layers. Such works are limited to [69, 70, 71]. SNAG [16] introduced application-awareness for data-intensive science and has been used to create novel NFV-based approaches to securing scientific data transfers [72].

In this work, we focus on optimizing service chain mapping across multiple data centers. Our work utilizes the NFV placement optimization strategies combined

with application awareness for effective SFC mapping.

3.8 Conclusions

In this work, we formulate the SFC mapping problem for multi-data center network topology and present an integer linear programming formulation. We propose a novel application-aware flow reduction (AAFR) algorithm which significantly reduces the flow processing costs across multiple data centers. While SFC-ILP provides the optimal SFC mapping, the proposed AAFR algorithm simplifies the SFC-ILP to reduce the flow-processing workloads for the mapping in the *service network*. We evaluate the performance of our AAFR algorithm and quantify the impacts of the number of SFCs and the SFC length on mapping cost, compare capacitated/uncapacitated cost gains, and finally investigate balancing flow-to-SFC mappings across data centers. Extensive performance evaluations show that our proposed AAFR algorithm reduces flow-processing costs by 70% for the capacitated SFC mapping case over the SFC-ILP. Further, for AAFR-U, our algorithm provides an additional 4.1% cost-gain over the AAFR-C case. We also demonstrate that our proposed AAFR algorithm is better at balancing flow-to-SFC mappings due to reduced flow-processing workloads.

Chapter 4

APRIL: An Application-Aware, Predictive and Intelligent Load Balancing Solution for Data-Intensive Science

4.1 Introduction

Recently, software defined networking (SDN) [73] and big data technologies [6] have received significant interest from both academia and industry. While big data, characterized by "5Vs" (volume, variety, velocity, value, and veracity), can have profound impacts on network design, such aspects have traditionally been addressed separately from the SDN paradigm. Some SDN features including control/data plane separation, programmability/reconfigurability, and logical centralization can positively benefit big data tasks such as data acquisition [74], delivery [75, 76, 77] and storage [78].

An ever-increasing need for big data in science has led to the rapid adoption of flexible (and programmable) high-speed network infrastructure. Such infrastructures typically rely on 100 Gbps links to support large-scale data movement. As an example, the high-energy physics community through the Large Hadron Collider (LHC) project, has experimental data transfers reaching tens of petabytes every year. Example data-intensive science workflows include the Compact Muon Solenoid (CMS) [7] and Laser Interferometer Gravitational-Wave Observatory

(LIGO) [24]. The most popular tools for big data movement include GridFTP [9] and XROOTD [10]. Since scientific research is highly data-driven, they place an undue burden on campus networks for data delivery, storage, and processing. Flexible and scalable end-to-end network architectures are necessary to ensure that data transfer applications use the network efficiently. Numerous scientific big data architectures have been developed (e.g. [79, 80, 13, 81]) to avoid performance hot-spots associated with traditional networks.

Numerous research efforts (e.g., [82, 83, 84]) have focused on SDN-based efficient network resource allocation algorithms and techniques for cloud and data center networks. However, most of these techniques target the optimization of network resources allocation based on factors such as traffic demand/loads, quality of service (QoS) requirements and usage patterns. Such key factors are generally highly volatile and time-varying in nature. Limited work has been done to model data transfers, or to predict the key factors that affect network resource allocation. Load balancing forms a critical component of big data network architectures as they directly influence application response times and maximize throughput via optimized traffic delivery to the application servers. Large-volume data transfers associated with big data provides many opportunities for understanding usage patterns and gain insights into network resource requirements. Rather than viewing big data systems as placing an undue burden on campus networks, we can exploit the insights gained in better understanding user/traffic demands. This results in optimized resource allocation to better serve the needs of campus network users.

In this work, we propose a novel intelligent load-balancing technique for improving server utilization using application-aware SDN and deep learning approaches. We also propose a deep learning approach for modeling large data

transfers in the campus network. Deep learning is a representational learning technique that can automatically discover data representations using a multi-layer network [85]. The representation is then used to infer large dataset information without the need for complex analysis. In this work, we demonstrate how deep learning based predictors can be utilized to make accurate predictions and forecast future network connections by relying on application-layer metadata. We implement two RNN variants for GridFTP connection time-series predictions. First, we explore a long short-term memory (LSTM) [86] based deep learning model for GridFTP predictive analytics. Next, we implement a gated recurrent unit (GRU) [87] based deep learning model. Our model incorporates an application-aware SDN system to classify traffic and to facilitate application-layer metadata exchange with the network-layer. We evaluate the efficacy of these models for both univariate and multivariate GridFTP datasets. To the best of our knowledge, this is the first effort to leverage application-aware SDN and deep learning techniques for modeling/predicting big data science data transfers for load balancing applications. We also show the effectiveness and superiority of our approach by evaluating it with a real-world dataset from a major U.S. CMS Tier-2 site. Further, we also present a deployment strategy to integrate APRIL predictive analytics, development of a GridFTP predictor model registry, service APIs for exposing the APRIL predictive service to external systems, and an approach to integrate APRIL for load balancing GridFTP servers.

4.1.1 Contributions and Organization

The specific contributions of this chapter are as follows:

1. *RNN-based Deep Learning Predictive Analytics*: We develop a long short-term

memory (LSTM) and a gated recurrent unit (GRU) based deep learning model for GridFTP connection time-series prediction. The model employs an application-aware SDN approach to obtain accurate and reliable traffic classification information to forecast future connections.

2. *Novel Application-aware Load Balancing*: We propose a novel application-aware, predictive and intelligent load balancing algorithm (APRIL). APRIL combines application-layer metadata with deep learning predictive analytics resulting in an intelligent load balancer.

3. *Real-world large-scale dataset*: We demonstrate our model's effectiveness through extensive evaluations using a real dataset from a U.S. CMS Tier-2 site. We present detailed data analysis to discover and identify long-term temporal dependencies in the dataset. We also compare our deep learning predictive model with other approaches such as Autoregressive integrated moving average (ARIMA) and multi-layer perceptron predictors.

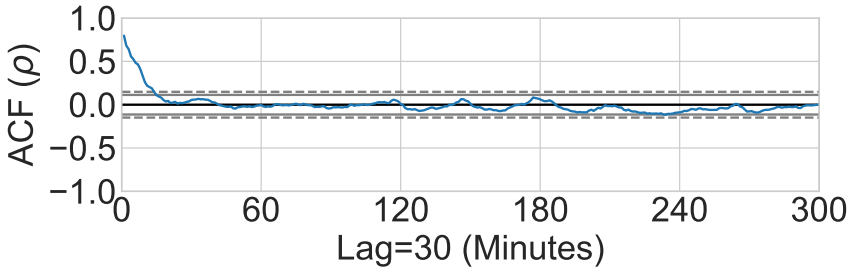
4. *Scalability and Improvements over LVS*: We also demonstrate the scalability of our solution by deploying our model on a project testbed network that has been set up to integrate with a U.S. CMS Tier-2 site. We compare the benefits and superiority of our solution with an existing production Linux Virtual Server (LVS) cluster.

5. *APRIL Deployment Strategy*: We present a deployment strategy to integrate APRIL predictive analytics module and the APRIL load balancer with the GridFTP ecosystem. We also present various deployment components, including the APRIL predictive analytics module, model registry and service APIs.

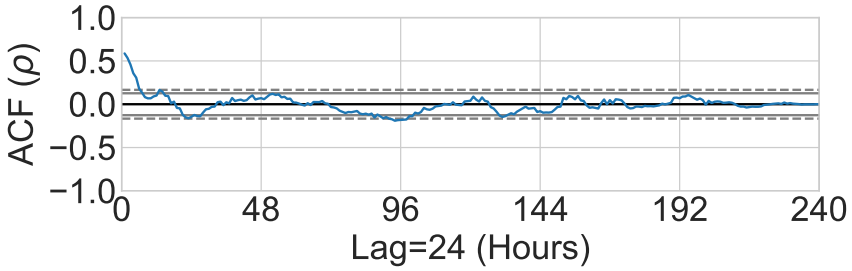
This chapter is structured as follows: Section 4.2 provides a brief overview of application-aware SDN in the context of load-balancing, the role of predictive analytics, and describe related works; Section 4.3 presents the exploratory analysis of our dataset; In Section 4.4, we detail our experimental network testbed, data management system and our experimental setup; Section 4.5 presents our univariate deep learning models and approaches to predicting GridFTP connection transfers. We describe and evaluate our Long Short-Term Memory and Gated Recurrent Unit (GRU) based load modeling and predictive analytics for forecasting GridFTP transfers across multiple servers. In Section 4.6, we present and evaluate the performance of multivariate RNN models using LSTM and GRU-based deep learning networks. The multivariate forecasting models predict GridFTP connections by taking a wider range of transfer parameters into account. In Section 4.7, we present our intelligent application-aware load-balancing solution (APRIL) for managing distributed high-throughput data transfers in the campus network. Section 4.8 presents the APRIL deployment strategy and integration with the GridFTP servers. We present the APRIL predictive analytics module workflow and pipelines, model registry and service API endpoints and integration with the GridFTP system. Lastly, in Section 4.9, we conclude our work and discuss the future work.

4.2 Related Work

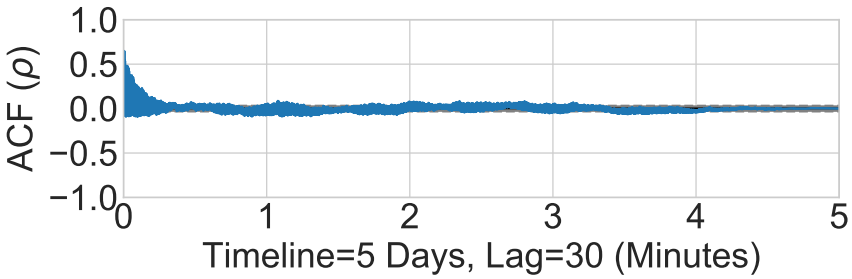
Numerous research efforts have focused on developing SDN load balancing mechanisms. The work in [88] presents a comprehensive survey on SDN load balancers. However, most of these are based on traffic routing mechanisms, or improve factors such as latency, synchronization, QoS, etc., or use heuristic optimizers to improve



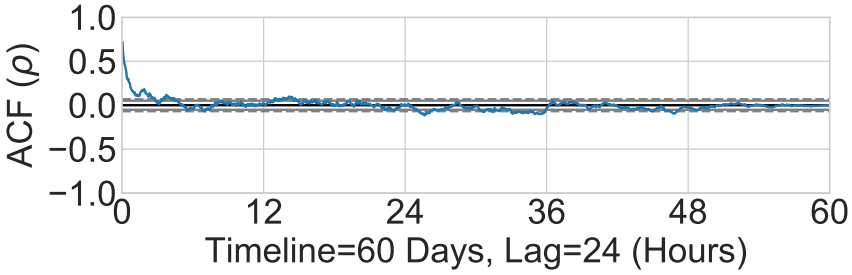
(a) Autocorrelation plot of $\mathbf{D}_m$, Duration= 300 minutes.



(b) Autocorrelation plot of $\mathbf{D}_h$, Duration= 240 hours.



(c) Autocorrelation plot of $\mathbf{D}_m$, Duration= 5 days.



(d) Autocorrelation plot of $\mathbf{D}_h$, Duration= 60 days.

Figure 4.1: Temporal autocorrelation properties of the datasets.

performance. Other works such as [89, 90] focus on LVS performance improvements. Although load balancers support specific transport-layer protocols, limited work has been done to develop true application-aware load balancing systems. Recent efforts such as [91, 92, 93] leverage machine/deep learning techniques for traffic classification and/or predictions. Integrated SDN and deep learning techniques have also be employed in VNF placement in NFV networks [94] and SDN security among others. A univariate autoregressive integrated moving average (ARIMA) based prediction framework was developed for forecasting GridFTP transfers in [95]. The model integrates with an application-aware SDN solution to preemptively drive network management decisions. However, the predictions are limited to a univariate dataset and the work does not address the scalability concerns associated with data-intensive science transfers. Different from the above, our work focuses on leveraging deep learning techniques to accurately model and predict the data-intensive science traffic load by exploiting an application-aware SDN solution. Further, we develop a novel intelligent load balancer that combines both application-layer metadata and future forecast knowledge to improve server utilization.

4.3 Data Analysis and Modeling

Data analysis and modeling is an essential step in providing us valuable insights about the temporal dependencies in the dataset. This information is critical to choosing an appropriate data and prediction model for improved forecast accuracy. In this section, we present our dataset used in the modeling and prediction. We also perform exploratory data analysis on the dataset to motivate our design choices.

4.3.1 Dataset

The dataset consists of GridFTP transfer connection data obtained from a major U.S. CMS Tier-2 site that performs frequent high-volume (low- and high-priority) transfers to Fermilab and holds over 3 petabytes of data. The site uses both the GridFTP protocol and XROOTd for bulk batch transfer jobs and interactive jobs, respectively. The data was obtained using an application-aware approach similar to the ones in [16, 17]. The dataset includes GridFTP connection information collected from a single U.S. CMS Tier-2 site over two years. The dataset represents over *800 million* GridFTP connections from both CMS and LIGO workflows. The dataset contains connection information classified by four CMS user roles (as defined in the CMS computing model [7]) and a single LIGO user role. A pool of twelve (12) GridFTP servers are employed by the site to serve both campus network users and external researchers. The four CMS user roles include: i) *US CMS Pool* representing analysis transfers associated with users' jobs, ii) *CMSProd* similar to (i), but representing production workflows, iii) *CMS PhEDEx* representing the CMS production data movement, iv) *LCG Admin* representing site availability monitoring transfers. The LIGO user role represents LIGO transfers that are opportunistic and share networking resources with CMS users. Other users include site administrators and computing center staff.

The GridFTP protocol uses encrypted control and data channels for data movement between end-points. Application-layer metadata is crucial for classifying the GridFTP connection information. Therefore, we rely on the GridFTP XIO plugin [96] to securely interface with the GridFTP servers and facilitate application metadata exchange with the SDN. The obtained connection information is pre-classified based on its workflow (or experiment) membership and also based on

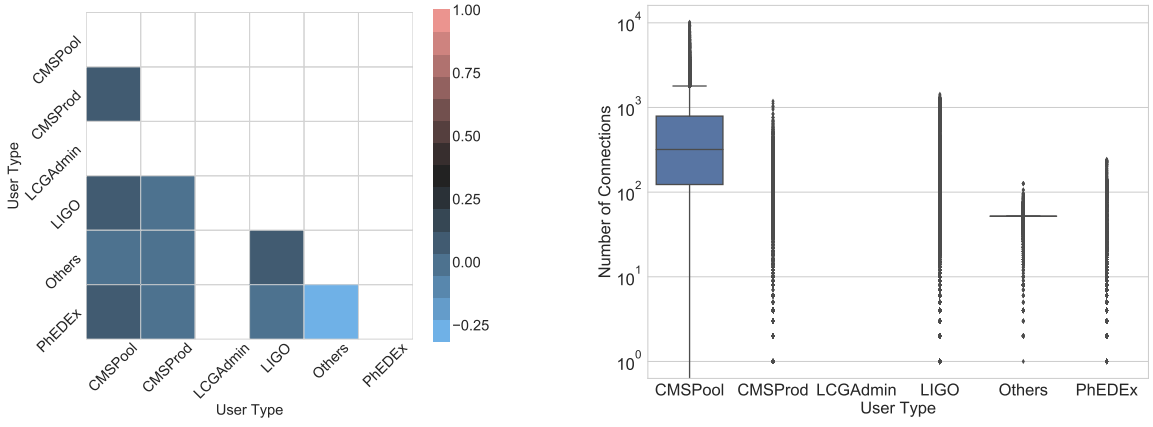
the user role within the workflow. User roles not belonging to either workflow are classified as "*Other*" users. The default dataset measurement granularity is in the order microseconds. However, as this granularity does not permit meaningful analysis, we use data aggregation to obtain aggregate statistics with granularities of one minute and one hour respectively. A *nonuple* (i.e., 9-tuple) is used to identify each connection uniquely and includes connection strings, user/workflow membership, files transferred, transfer direction and a status field. We denote the GridFTP connection dataset by $\mathbf{G} = \{g_{(a_1,\cdots,a_9),n,t}\}, \forall t, \forall n \subset N$. We denote the downlink and uplink connections by $\mathbf{D} = \{d_{(a_1,\cdots,a_9),t}\}, \forall t, \forall n \subset N$ and $\mathbf{U} = \{u_{(a_1,\cdots,a_9),t}\}, \forall t, \forall n \subset N$, respectively, for N observations. We normalize both datasets $\mathbf{D}$ and $\mathbf{U}$ within a range of $[0, 1]$. To achieve this, we use *Min-Max* scaling to compute the normalized values $\hat{\mathbf{G}}$ as:

$$\hat{\mathbf{G}} = \frac{\mathbf{G} - \mathbf{G}_{min}}{\mathbf{G}_{max} - \mathbf{G}_{min}} \tag{4.1}$$

where, $\mathbf{G}_{max}$ and $\mathbf{G}_{min}$ represent the maximum and minimum values of the set $\mathbf{G}$, respectively. Also, we note that $\mathbf{D} \subset \mathbf{G}$ and $\mathbf{U} \subset \mathbf{G}$. Lastly, we denote the datasets with different measurement granularities by $\mathbf{G}_m$ (one minute) and $\mathbf{G}_h$ (one hour), respectively.

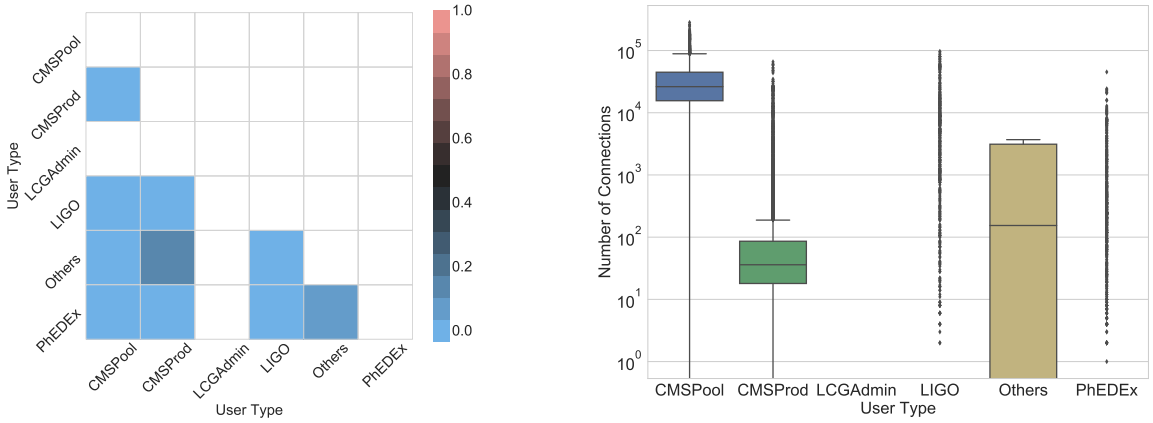
4.3.2 Exploratory Analysis

The objective of our exploratory data analysis is to discover and identify data dependencies in $\mathbf{G}$, $\mathbf{D}$ and $\mathbf{U}$ in the temporal domain. Initially, we analyze the dataset for the presence of systematic patterns combined with random error. By identifying and removing non-stationary processes within the dataset, we can obtain a dataset with independent identically distributed (i.i.d.) components that



(a) Correlation Heatmap – $\mathbf{D}_m$.

(b) $\mathbf{D}_m$ user roles.



(c) Correlation Heatmap – $\mathbf{D}_h$.

(d) $\mathbf{D}_h$ User roles.

Figure 4.2: Short- and long-run dataset properties and cross-correlation.

are amenable to modeling using linear regression on exogenous variables. We also examine temporal autocorrelation and data dependency between different user roles for both datasets, i.e., **D** and **U**, respectively. We make the following important observations:

Observation 1: *The dataset* **G** *exhibits non-stationarity.*

We denote the k^{th} observation as $G_k = g_{(a_1,\cdots,a_9),k,t}$. The dataset **G** is strictly

stationary if:

$$(G_1, G_2, \cdots, G_n)' \stackrel{\mathrm{d}}{=} (G_{1+h}, G_{2+h}, \cdots, G_{n+h})', \forall n \geq 1 \tag{4.2}$$

where, $\stackrel{\mathrm{d}}{=}$ denotes that the two random vectors share the same joint distribution function. The Augumented Dickey-Fuller (ADF) test is a widely used tool to test stationarity [97]. The autocorrelation plots of the two downlink datasets with measurement granularities of one minute and one hour are shown in the Figure 4.1. Figures 4.1a and 4.1b represent the short-run dataset (measured over 300 minutes and 240 hours, respectively), while Figures 4.1c and 4.1d represent the corresponding long-run datasets (measured over 5 and 60 days, respectively). The presence of significant autocorrelation in the lags for time $t > 1$ is an indicator of non-stationarity. This is also verified by running the ADF test on the above datasets, resulting in the test accepting the null hypothesis, indicating a non-stationary process. Thus, (first-order) differencing is required to stationarize the datasets.

Observation 2: *The dataset* **G** *has non-zero temporal autocorrelation properties.*

In the absence of significant autocorrelation, the data points in Figure 4.1 fall within the confidence interval bands represented by the dashed lines. The sample autocorrelation function (ACF) is commonly used to identify and discover data dependency between the observations. We define the sample ACF for the dataset **G** by:

$$\rho(h) = \frac{\hat{\gamma}(h)}{\hat{\gamma}(0)} = \frac{\sum_{t=1}^{n-|h|}(G_{t+|h|} - \bar{G})(G_t - \bar{G})}{\sum_{t=1}^{T}(G_t - \bar{G})^2}, -n < h < n \tag{4.3}$$

where, $\bar{G}$ represents the sample mean of $G_1, G_2, \cdots, G_n$, and n is the total number of observations in the sample. The autocorrelation plots shown in Figure 4.1

exhibits high values for ρ for lags $h > 1$. This indicates the presence of systematic patterns mixed with random errors.

Observation 3: *The dataset* $\mathbf{G}$ *exhibits low cross-correlation in the temporal domain, between different user roles.*

We use the correlation matrix to assess the strength of the relationship between two user roles with either the same, or, with different workflow memberships. Figure 4.2 shows the correlation matrix heatmaps for the short-run datasets (both 1m and 1h aggregates) along with the corresponding box-whiskers plots. Only the lower triangular correlation matrix is presented in the correlation heatmaps for brevity. As shown in the Figures 4.2a and 4.2c, we observe low to moderate cross correlation between the users' transfer connections for both datasets. For the short-run dataset with one-minute aggregates $\mathbf{D}_m$, we see low- to moderate positive correlation between different users' transfers and low negative correlation in one instance. Similarly, we see low positive correlation between the users' transfers for the short-run dataset with hourly aggregates $\mathbf{D}_h$. Both correlation matrices represent the users' connection transfer relationships, and provide some insight into the number of parameters required to estimate them. The box plots for datasets $\mathbf{D}_m$ and $\mathbf{D}_h$ are shown in Figures 4.2b and 4.2d, respectively. From the box plot, we observe that the *CMSPool* users exhibit large variations across both $\mathbf{D}_m$ and $\mathbf{D}_h$ datasets. *CMSProd* and *Other* show large hourly variations, while *LIGO* and *PhEDEx* exhibit little variation across datasets but have significant outliers. The box plots are useful in helping us understand the distribution characteristics of the datasets and in outlier detection.

4.4 Experimental Testbed

In this section, we present our experimental setup, an application-aware architecture to integrate with the GridFTP server pool, our data management framework, the testbed network topology, and how it interfaces with the Linux Virtual Server (LVS) [98] load balancing cluster. Our experimental network topology is shown in Figure 4.3. It consists of five components namely: (i) the GridFTP server pool, (ii) the LVS load balancing cluster and a LVS redirector, both of which are transparent to end-user applications, (iii) the application-aware SDN infrastructure, (iv) the Elastic stack cluster for data management, and (v) the SDN data plane infrastructure and 100 Gbps connectivity to the wide area network (WAN).

4.4.1 Application-aware SDN and GridFTP Integration

Application-awareness is achieved using the Globus eXtensible I/O (XIO) [96] extensible I/O library. We develop a Globus XIO SDN Callout to interface with the SDN infrastructure. The XIO Callout module integrates GridFTP servers with the SDN via an SDN application similar to SNAG [16]. It also uses a Hadoop Distributed File System (HDFS) plugin to interact with GridFTP servers' distributed storage/processing infrastructure. The XIO Callout module facilitates the exchange of application-layer metadata with the SDN infrastructure.

4.4.2 Network Testbed Topology

Our network testbed setup is an exemplary implementation of an SDN that can handle frequent, high-volume, low- and high-priority data transfers from a major U.S. CMS Tier-2 site to Fermilab. The U.S. CMS Tier-2 site holds over 3 PB of data, and uses both GridFTP and XROOTD protocols for bulk batch transfer jobs and interactive jobs, respectively. Our testbed network architecture

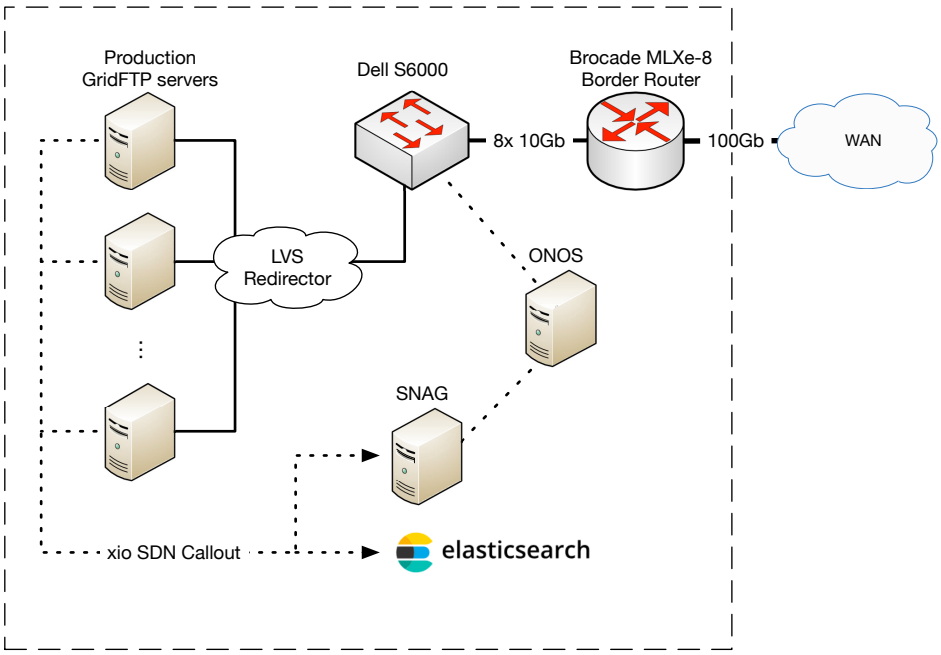


Figure 4.3: Experimental Testbed.

effectively combines several important components including: (i) Intelligent flow control, flow forwarding and management using the ONOS SDN controller, (ii) An application-aware SDN application to facilitate secure exchange of application-layer metadata with the network-layer, (iii) A GridFTP Callout module that serves as an interface between the GridFTP servers, its HDFS storage backends and the SDN infrastructure. The XIO callout communicates with SDN controller using a secure representational state transfer (REST) application programming interface (API), and (iv) A Brocade MLXe border router at the campus network edge with 100 Gbps WAN connectivity to Internet2. A Dell S6000 40 GbE switch to serve as the CMS cluster network core hosting 12 production GridFTP servers and an Edge-Core AS4600-54T SDN-capable switch for testing purposes.

4.4.3 Data Management System

Our current dataset includes information of over *800 million* GridFTP transfer connections from both CMS and LIGO workflows. This dataset is consistently growing as it is updated with new real-time connection information, while also being expanded to other workflows. To manage this large dataset, we employ a 11-node Elastic stack [43] cluster with the following configuration: (i) Master Nodes: 3×Dell SC1435, 16GB RAM, 250GB HDDs, (ii)Hot-Data/Ingest Nodes: 3×Sun SunFire X2200, 32GB RAM, 240GB SSDs, (iii) Warm Data Nodes: 5×Sun SunFire X2200, 32GB RAM, 2TB HDDs, and (iv) 1Gb Ethernet interconnects between all nodes.

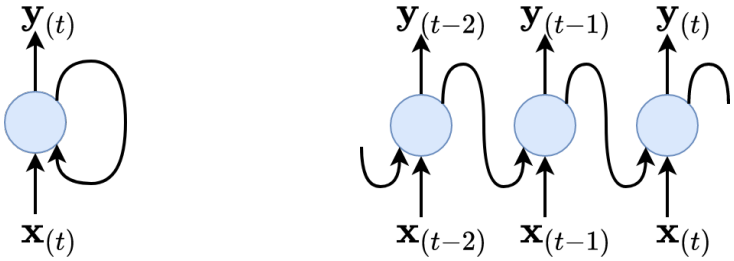
This Elastic cluster is responsible for storing all application-aware information exchanged between the GridFTP server pool and the SDN infrastructure. A *syslog* style file on each of the 12 GridFTP servers feeds a *filebeat* agent (a lightweight data shipper for the Elastic stack), which in turn feeds the *logstash*, a server-side data ingestion pipeline on the Elastic cluster.

4.5 Univariate Load Modeling and Predictive Analytics

4.5.1 Overview

We propose the use of recurrent neural networks (RNNs) for modeling and predicting time-series data. RNNs are a class of generalized feed-forward neural networks that exhibit dynamic temporal behavior and can, therefore, be used for time sequence modeling. The RNN can maintain and use internal states (memory) to process input sequences. However, standard RNNs suffer from well-known problems of vanishing/exploding gradients and therefore, using RNNs to model long-term dependencies is difficult [99]. Many solutions have been

proposed including long short-term memory (LSTM) [100] and gated recurrent units (GRU) [87] to capture and model long-term temporal dependencies [101]. Both LSTM and GRU use "forget" gates that enable a model to both learn to forget previous states (i.e., dropping memory), and to update current states (i.e., adding new memory). In this section, we present the RNN univariate models used to predict GridFTP connection transfers classified by user type and transfer direction.



(a) Single RNN Neuron. (b) An Unrolled RNN Network.

Figure 4.4: RNN Network Structure.

A recurrent neural network (shown in Figure 4.4) is similar to a feedforward neural network with a significant difference, namely, a backward-pointing connection with each neuron feeding its output back to itself. Figure 4.4a shows a single RNN neuron. Figure 4.4b shows an unrolled RNN network, with each neuron receiving input and its output from a previous time step. During a time step t, each recurrent neuron processes $\mathbf{x}_{(t)}$, an input vector, and $\mathbf{y}_{(t-1)}$, an output vector from the previous time step (i.e. $t-1$). Thus, each recurrent neuron has two weight vectors $\mathbf{w}_{(x)}$ and $\mathbf{w}_{(y)}$. For a single recurrent layer, we can combine $\mathbf{w}_{(x)}$ and $\mathbf{w}_{(y)}$ from each neuron to form $\mathbf{W}_{(x)}$ and $\mathbf{W}_{(y)}$ weight matrices, respectively. Given a bias vector $\mathbf{b}$ and an activation function $\phi(\cdot)$, the output of a recurrent layer is given by:

$$\mathbf{y}_{(t)} = \phi(\mathbf{W}_x^T \cdot \mathbf{x}_{(t)} + \mathbf{W}_y^T \cdot \mathbf{y}_{(t-1)} + \mathbf{b}) \tag{4.4}$$

As each recurrent neuron's output is a function of all inputs from previous steps,

the neurons are also referred to as memory cells. We denote the recurrent cells' state at time t by $\mathbf{h}_{(t)}$, where $\mathbf{h}_{(t)} = f(\mathbf{x}_{(t)}, \mathbf{h}_{(t-1)})$.

Recurrent neural networks (RNN) suffer from numerous problems, including unstable gradients problem and short-term memory problems. Works such as [102] and [103] have explored normalization techniques to solve the unstable gradient problem with limited gains. As data transformations during RNN traversals lead to information loss at each time step, long-term memory cells were proposed. One of the most popular long-term memory cell solutions is the long short-term memory (LSTM) [86].

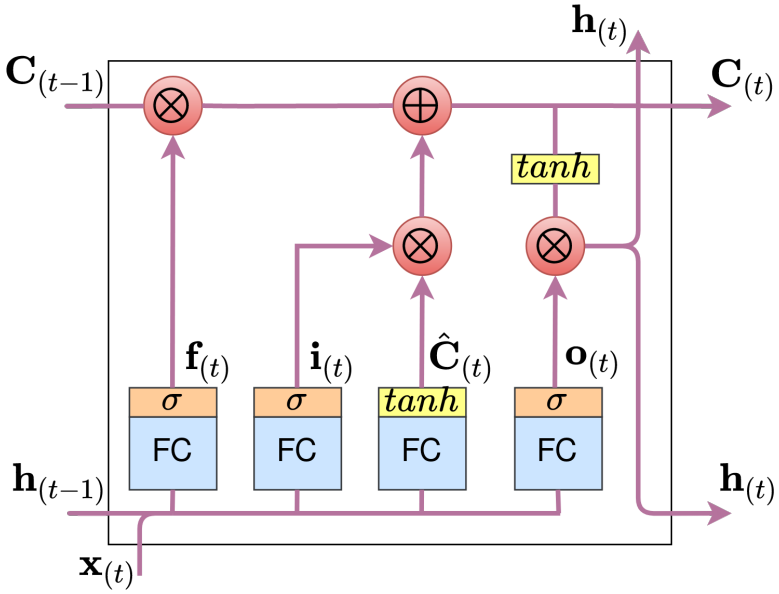


Figure 4.5: Long Short-Term Memory (LSTM) Cell.

An LSTM cell is shown in Figure 4.5. Unlike an RNN cell, an LSTM cell state is divided into a short-term state, $\mathbf{h}_{(t)}$, and a long-term state, $\mathbf{C}_{(t)}$. As shown in Figure 4.5, the long-term state $\mathbf{C}_{(t-1)}$ first traverses through a forget gate to drop some prior memory. It then adds new memories selected by the input $\tilde{\mathbf{C}}_{(t)}$. The input vector $\mathbf{x}_{(t)}$ and the short-term state vector $\mathbf{h}_{(t-1)}$ are fed to four fully-connected (FC) units. The outputs of three fully-connected units form gate

controllers, namely: (i) the forget gate $\mathbf{f}_{(t)}$, (ii) the input gate $\tilde{\mathbf{C}}_{(t)}$, and (iii) the output gate $\mathbf{o}_{(t)}$. The output of the second FC unit $\mathbf{g}_{(t)}$ contributes to the outputs $\mathbf{C}_{(t)}$ and $\mathbf{h}_{(t)}$.

The LSTM cell state computations are summarized below:

$$\mathbf{f}_{(t)} = \sigma(\mathbf{W}_{xf}^T \cdot \mathbf{x}_{(t)} + \mathbf{W}_{hf}^T \cdot \mathbf{h}_{(t-1)} + \mathbf{b}_f) \tag{4.5}$$

$$\mathbf{i}_{(t)} = \sigma(\mathbf{W}_{xi}^T \cdot \mathbf{x}_{(t)} + \mathbf{W}_{hi}^T \cdot \mathbf{h}_{(t-1)} + \mathbf{b}_i) \tag{4.6}$$

$$\mathbf{o}_{(t)} = \sigma(\mathbf{W}_{xo}^T \cdot \mathbf{x}_{(t)} + \mathbf{W}_{ho}^T \cdot \mathbf{h}_{(t-1)} + \mathbf{b}_o) \tag{4.7}$$

$$\tilde{\mathbf{C}}_{(t)} = tanh(\mathbf{W}_{x\tilde{C}}^T \cdot \mathbf{x}_{(t)} + \mathbf{W}_{h\tilde{C}}^T \cdot \mathbf{h}_{(t-1)} + \mathbf{b}_{\tilde{C}}) \tag{4.8}$$

$$\mathbf{C}_{(t)} = \mathbf{f}_{(t)} \otimes \mathbf{C}_{(t-1)} + \mathbf{i}_{(t)}) \otimes \tilde{\mathbf{C}}_{(t)} \tag{4.9}$$

$$\mathbf{C}_{(t)} = \mathbf{f}_{(t)} \otimes \mathbf{C}_{(t-1)} + \mathbf{i}_{(t)}) \otimes \tilde{\mathbf{C}}_{(t)} \tag{4.10}$$

In the above equations, the matrices $\mathbf{W}_{xf}^T, \mathbf{W}_{xi}^T, \mathbf{W}_{x\tilde{C}'}^T, \mathbf{W}_{xo}^T$ represent the $\mathbf{x}_{(t)}$ (input vector) connection weights. The matrices $\mathbf{W}_{hf}^T, \mathbf{W}_{hi}^T, \mathbf{W}_{h\tilde{C}}^T, \mathbf{W}_{ho}^T$ represent the $\mathbf{h}_{(t)}$ (short-term state vector) connection weights. Lastly, the terms $\mathbf{b}_f, \mathbf{b}_i, \mathbf{b}_o, \mathbf{b}_{\tilde{C}}$ represent the bias terms for each layer. We use a deep LSTM network to make GridFTP connection predictions. Additional details of the deep LSTM network are provided in Section 4.5.2.

We also evaluate the use of a deep GRU network to make GridFTP connection predictions. A GRU cell is shown in Figure 4.6. The GRU cell is a simplified version of LSTM cell, but is known exhibit similar performance [104].

Unlike LSTM, both state vectors in GRU are merged into a single vector $\mathbf{h}_{(t)}$.

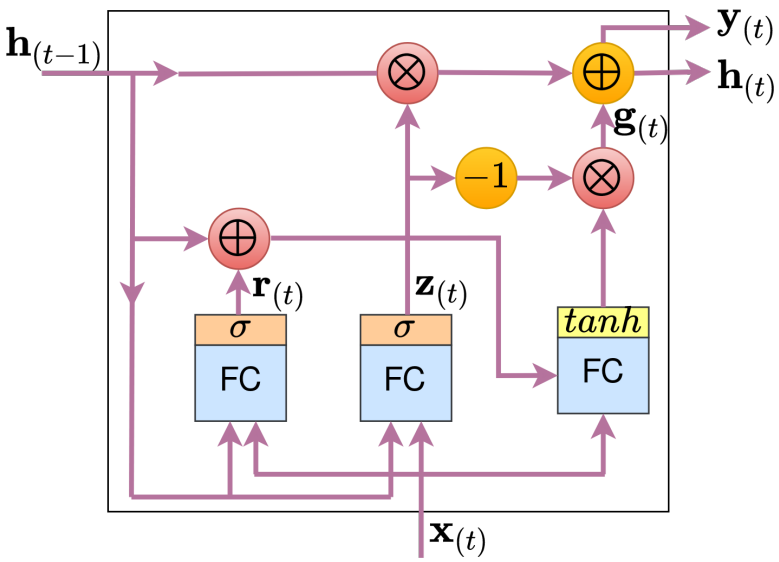


Figure 4.6: Gated Recurrent Unit (GRU) Cell.

The GRU cell state computations are summarized below:

$$\mathbf{z}_{(t)} = \sigma(\mathbf{W}_{xz}^T \cdot \mathbf{x}_{(t)} + \mathbf{W}_{hz}^T \cdot \mathbf{h}_{(t-1)} + \mathbf{b}_z) \tag{4.11}$$

$$\mathbf{r}_{(t)} = \sigma(\mathbf{W}_{xr}^T \cdot \mathbf{x}_{(t)} + \mathbf{W}_{hr}^T \cdot \mathbf{h}_{(t-1)} + \mathbf{b}_r) \tag{4.12}$$

$$\mathbf{g}_{(t)} = tanh(\mathbf{W}_{xg}^T \cdot \mathbf{x}_{(t)} + \mathbf{W}_{hg}^T \cdot (\mathbf{r}_{(t)} \otimes \mathbf{h}_{(t-1)}) + \mathbf{b}_g) \tag{4.13}$$

$$\mathbf{h}_{(t)} = \mathbf{z}_{(t)} \otimes \mathbf{h}_{(t-1)} + (1 - \mathbf{z}_{(t)}) \otimes \mathbf{g}_{(t)} \tag{4.14}$$

The terms $\mathbf{W}$ and $\mathbf{b}$ denote the weight matrix and the bias terms, respectively. Two types of activation functions are used by the fully-connected (FC) units namely: (i) $\sigma(\cdot)$, which is the sigmoid activation function, and (ii) $tanh(\cdot) = 2\sigma \cdot 2(x) - 1$, the hyperbolic tangent function. The terms $\oplus$ and $\otimes$ denote the sum and dot products, respectively. The update gate $\mathbf{z}_{(t)}$ helps the model control the amount of historical information passed to the next state. A reset gate $\mathbf{r}_{(t)}$ controls the amount of past information to forget. We note that a single gate controller is used to control both

the update gate and the input gate. Whenever a new memory must be stored, its (storage) location is erased first. Lastly, in comparison to an LSTM cell, we also note the absence of an output gate, and a full state vector is an output at every time-step.

	Univariate Models			Multivariate Models		
One	**MLP**	**GRU**	**LSTM**	**MLP**	**GRU**	**LSTM**
Epochs	50			50		
Input Scaling	Min-Max			Min-Max, Standard		
Input Aggregation	per-minute (5 days), per-hour (60 days)			per-minute (7 days)		
Batch Size	16			64		
Features	File size			File size, user-role, direction and locality		
H1 Parameters	512	12864	17152	320	13440	17920
H2 Parameters	2080	9408	12544	2080	9408	12544
H3 Parameters	528	2400	3200	528	2400	3200
Output Layer	17	17	17	17	17	17
Total Parameters	3,137	24,689	32,913	2,945	25,265	33,681
Train/Validation Split	80/20			80/20		

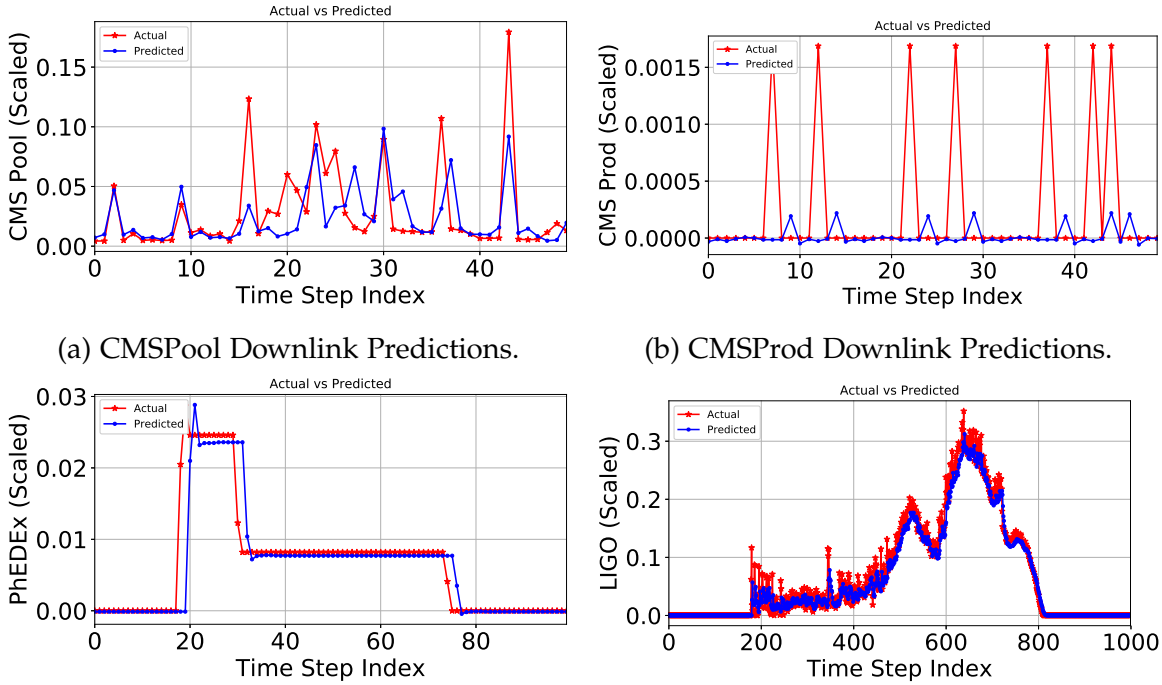
Table 4.1: Univariate and Multivariate Deep Learning Model Parameters.

4.5.2 Temporal Prediction Model

Taking the observations in Section 4.3.2 into consideration, we develop both LSTM and GRU-based deep learning networks to model and predict the temporal GridFTP connection information classified by user role. In this section, we focus on developing univariate models to predict individual user behavior for a given transfer direction. These deep learning networks will forecast per-role connections for both CMS and LIGO users. We build separate models for each user role, and the predictive analytics system chooses the best model on a per-role basis to make effective forecasts. In the case of an LSTM memory cell, both short-term ($\mathbf{h}_{(t)}$) and long-term ($\mathbf{C}_{(t)}$) states are updated based on combining the input state with the past state. We describe this process in Equations 4.5–4.10. The GRU memory cell updates the current hidden state $\mathbf{h}_{(t)}$ by combining the input and the past state as described in Equations 4.11–4.14. To predict the future value $G_{k,t+1} = g_{(a_1),k,t+1}$, we rely on past T observations, i.e. $\sum_{\tau=t-T}^{t} G_{k,\tau}$. For both networks (LSTM and GRU), a 3-layer deep RNN with 64, 32 and 16 cells is used for forecasting per-role future connection values. We use a step-size of seven (7) at the input layer. The input is normalized using a min-max scaling defined in Equation 4.1. A dropout layer is added to final hidden layer with a probability of 50% to avoid overfitting. We also note that Adam optimizer [105] was used with a training batch size of 16, with a linear activation function at the output layer. A detailed list of the deep learning models' parameters are presented in Table 4.1.

4.5.3 Performance Evaluation

We compared our deep RNN models with two other time series analysis and prediction methods. First, we compare the prediction capabilities of our models



(a) CMSPool Downlink Predictions.

(b) CMSProd Downlink Predictions.

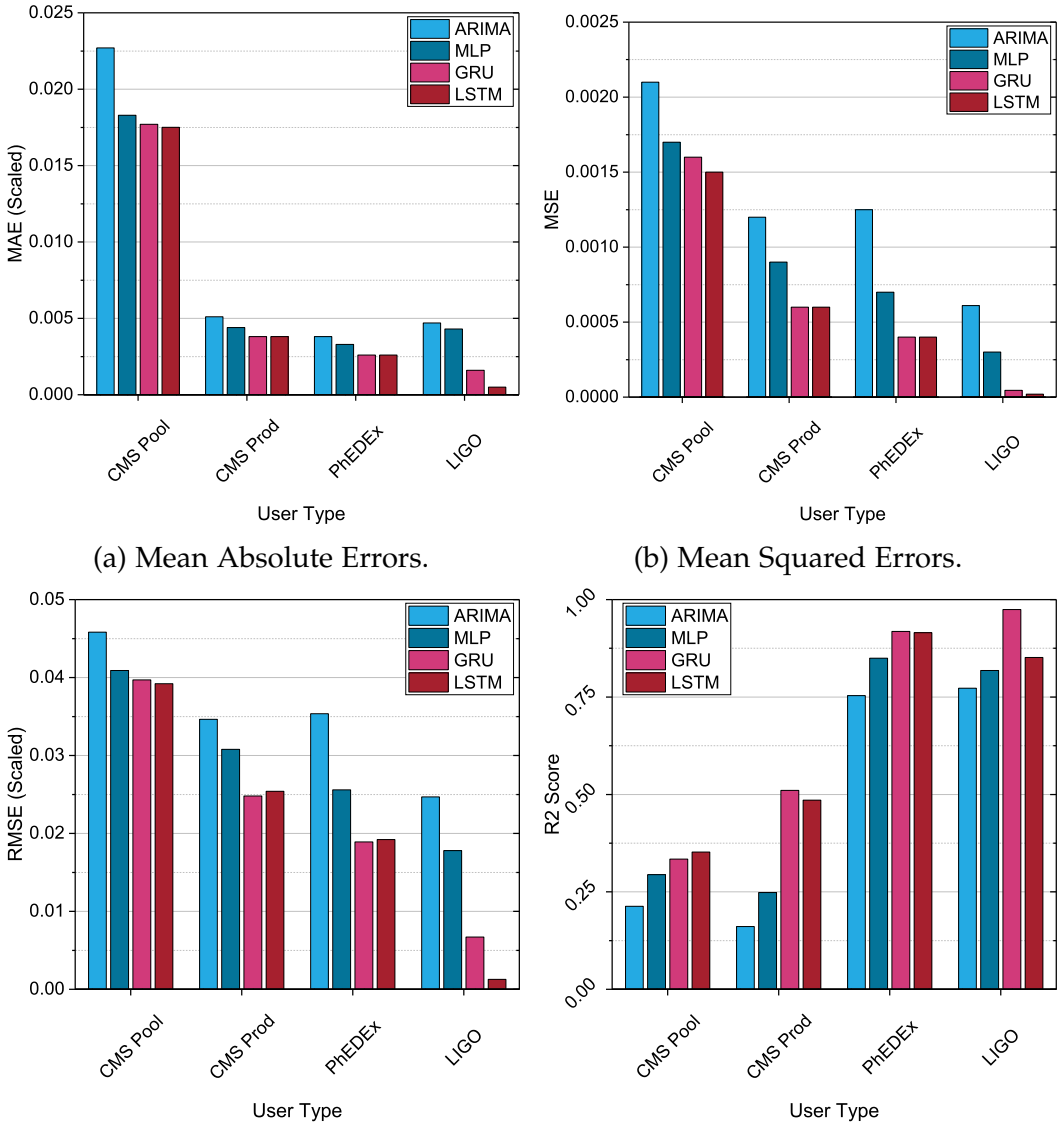
(c) CMS PhEDEx Downlink Predictions.

(d) LIGO Downlink Predictions.

Figure 4.7: Predicted values vs. Actual $\mathbf{D}_m$ measurements by user role.

with an Autoregressive integrated moving average (ARIMA) [106] multi-step predictor. ARIMA is a widely used method for time-series analysis and forecasting [106]. An ARIMA model is selected by minimizing the model's Akaike Information Criterion (AIC). The ARIMA model has three parameters: the AR model order p, the MA model order q, and the differencing component d. The model parameters (p,d,q) search-space is upper-bounded by (10,2,10). Next, we compare our models with a deep multi-layer perceptron (MLP) predictor consisting of three dense hidden layer with 64, 32, and 16 fully connected units, a dropout layer added to the final hidden layer with a probability of 50%, hyperbolic tangent activation functions at the hidden layer and linear activation at the output layer.

We compare the performance of our models with the ARIMA and the MLP predictors using three widely used performance metrics namely: (i) Mean Ab-



(a) Mean Absolute Errors.

(b) Mean Squared Errors.

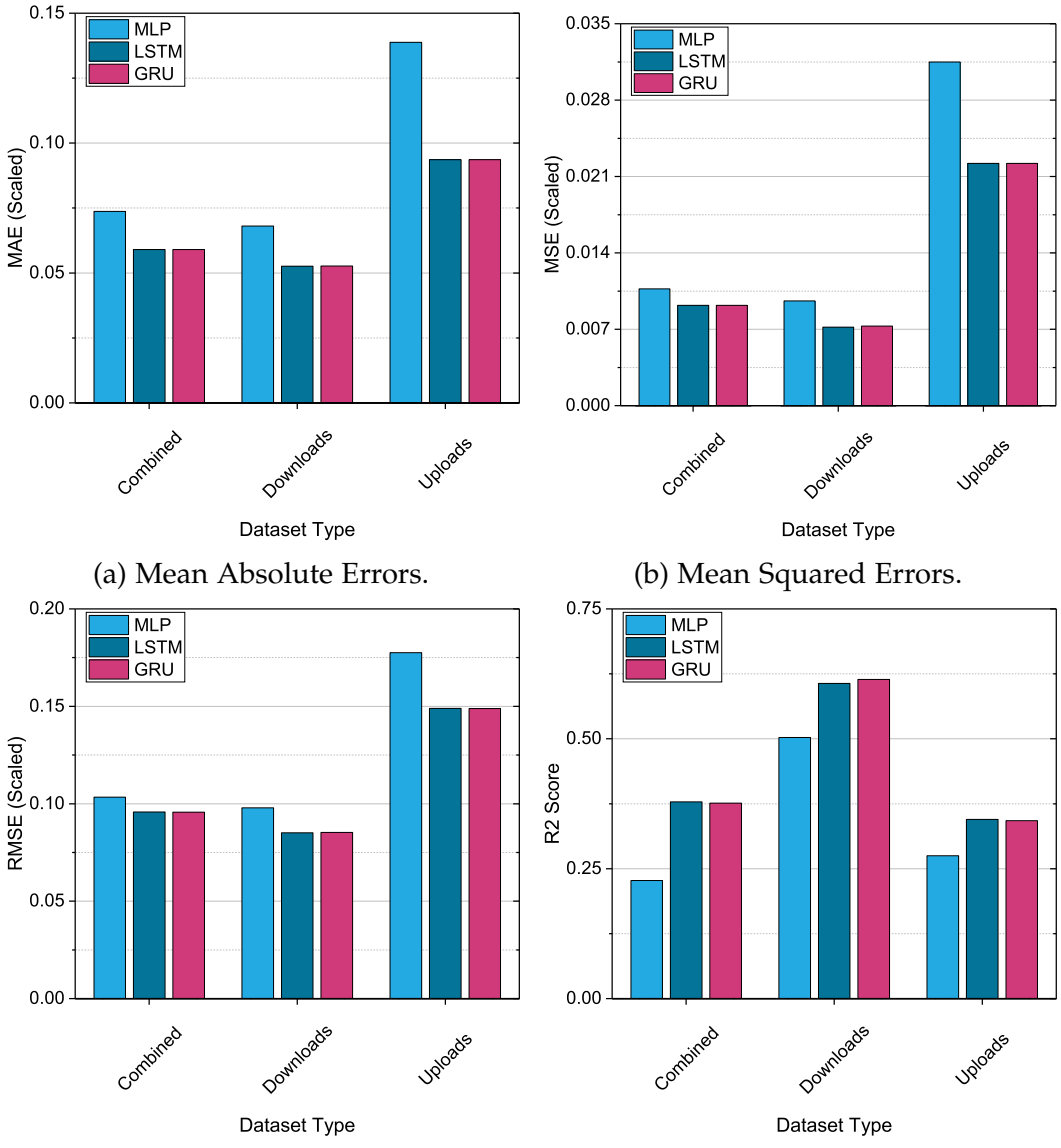
(c) Root Mean Squared Errors.

(d) Coefficient of Determination.

Figure 4.8: Univariate prediction model performance categorized by user role.

solute Error (MAE), (ii) Mean Squared Error (MSE), and (iii) the coefficient of determination (r^2 score). We also present the root mean squared error (RMSE) metric measurements for convenience. The dataset used in making the predictions $\mathbf{G}_m$ (1-minute aggregate granularity), was measured over 5 days and contained over 512,000 GridFTP transfer connection records from six user roles described

in Section 4.3.1. Our dataset is initially classified by data transfer direction, i.e., uploads and downloads. This dataset is further partitioned into a training set and a validation set, categorized by user roles. Next, we present the prediction results of our univariate deep RNN models and compare it with ARIMA and MLP forecasting models.



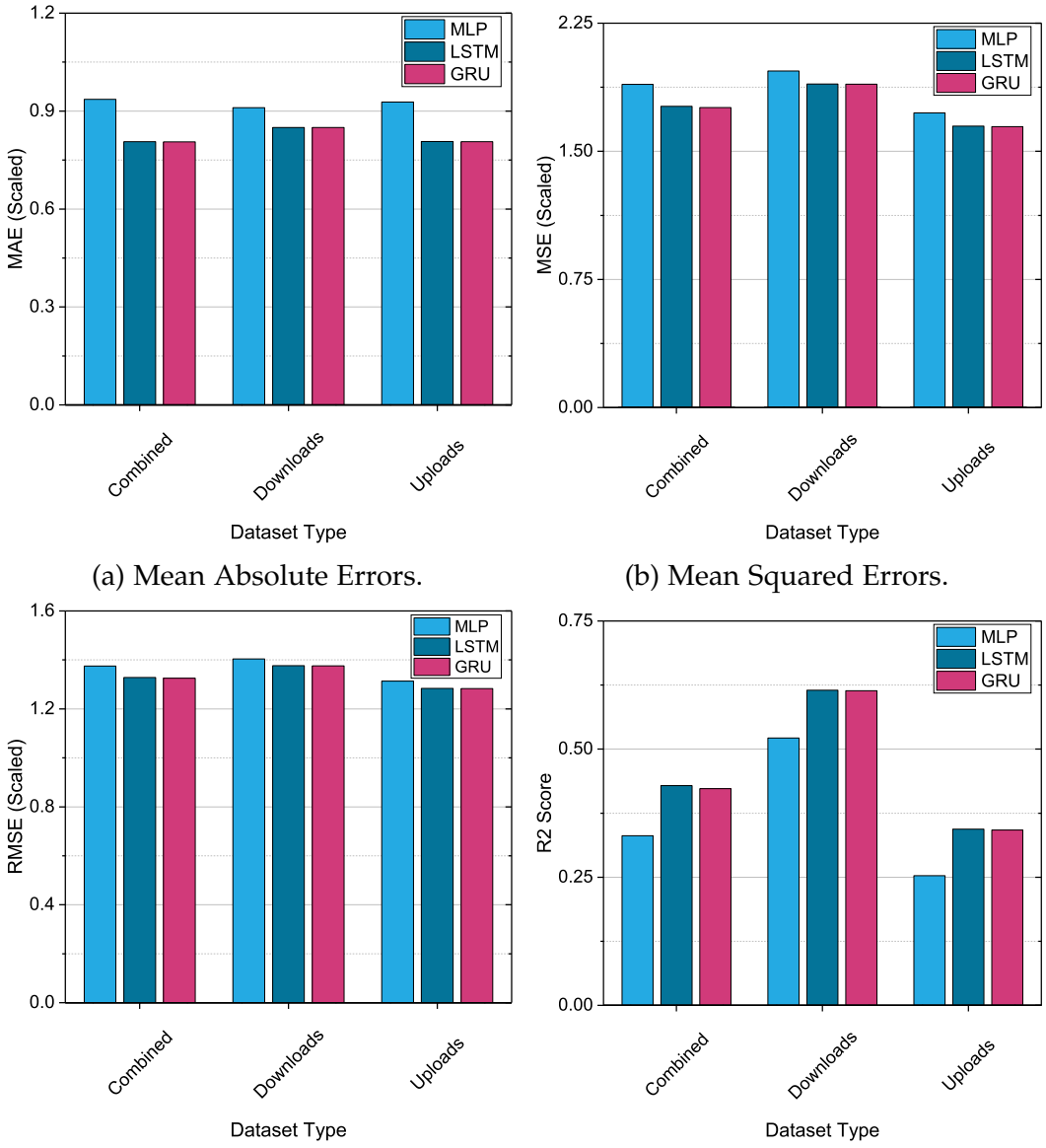
(a) Mean Absolute Errors.

(b) Mean Squared Errors.

(c) Root Mean Squared Errors.

(d) Coefficient of Determination.

Figure 4.9: Multivariate prediction model performance categorized by dataset (min-max scaling).



(a) Mean Absolute Errors.

(b) Mean Squared Errors.

(c) Root Mean Squared Errors.

(d) Coefficient of Determination.

Figure 4.10: Multivariate prediction model performance categorized by dataset (standard scaling).

4.5.4 Prediction Results and Discussion

The prediction results for the $\mathbf{D}_m$ dataset (downlink connections 1-minute aggregate granularity) is shown in Figure 4.7. Figures 4.7a, 4.7b, 4.7c and 4.7d show the

actual vs. predicted connection values for US CMS Pool, CMS Prod, CMS PhEDEx and LIGO users, respectively. From the results, we see that predicted results show a good fit with actual observations. The prediction models' performance categorized by user role is presented in Figure 4.8. Specifically, we present the MAE, MSE, RMSE and r^2 scores in the Figures 4.8a, 4.8b, 4.8c, and 4.8d, respectively. First, we compare the prediction performance gains of the GRU model over ARIMA and MLP models. Our proposed GRU model, depending on the user role, shows an improved error performance between 22.03%–65.96%, 23.8%–92.6%, and 13.37%–72.87% regarding MAE, MSE and RMSE, respectively over the ARIMA model. Our model also shows r^2 score improvements between 21.8%–217.14% over the ARIMA model. Further, our model, in comparison to the MLP model, shows an improved error performance between 3.28%–62.8%, 5.88%–85%, and 2.93%–62.36% regarding MAE, MSE and RMSE, respectively. It also shows r^2 score improvements between 8.06%–105.64% over the MLP model.

Next, we compare the LSTM model performance over ARIMA and MLP. Depending on the user role, our LSTM model shows error performance improve between 22.91%–98.94%, 28.57%–99.99%, and 14.46%–99.48% regarding MAE, MSE and RMSE, respectively over the ARIMA model. Our LSTM model also shows r^2 score improvements between 10.27%–201.49% over the ARIMA model. Further, in comparison to the MLP model, our LSTM model shows an improved error performance between 4.37%–98.8%, 11.76%–99.99%, and 4.15%–99.28% regarding MAE, MSE and RMSE, respectively. Our LSTM model also shows r^2 score improvements between 4.09%–95.49% over the MLP model.

When comparing LSTM vs. GRU, we observe that both models perform similarly. The models' error performance is within 1.13%, 6.25% and 2.36% of each other regarding MAE, MSE and RMSE, respectively. A significant difference

between the models is the total number of network training parameters outlined in Table 4.1, which leads to increased training times for the LSTM network. The above results show the effectiveness of our GRU-based deep RNN model in making accurate GridFTP connection load predictions. Also, importantly, the superiority of our design ensures that it takes long-term temporal dependencies into account. Such a system is vital in providing timely intelligence to load balancing systems. In the following section, we develop and evaluate multivariate prediction models for GridFTP connection transfers by combining user roles, transfer directions, transfer file sizes, and server locality into account.

4.6 Multivariate Load Modeling and Predictive Analytics

4.6.1 Overview

Thus far, we have explored univariate models for GridFTP predictive analytics. In this section, we explore multivariate deep learning models that employ a comprehensive range of GridFTP connection parameters to forecast future connections. Notably, we evaluate model learning based on a multivariate GridFTP dataset consisting of a few new parameters including transfer file size, user role, transfer direction and server locality. These additional parameters provide further context to model and can help improve predictive analytics.

We extend the recurrent neural networks (RNNs) described in the previous section to model and forecast multivariate GridFTP datasets. Our multivariate dataset consists of GridFTP transfers measured over seven days, aggregated into one-minute intervals. We first classify these aggregates by user role, then by transfer direction and finally by server locality. The goal is to forecast data transfer sizes by user roles for each direction and by server locality. At each time step,

the RNN processes a 5-tuple consisting of connection timestamp, file size, user role, direction and server locality. Further, we also explore model performance by limiting the dataset to a single transfer direction. In this case, our dataset is composed of a 4-tuple for each direction. As our dataset contains categorical information, we use encoding techniques to convert the categorical feature variables to numeric features. We evaluate three sets of models for forecasting performance: (i) multivariate dataset with both transfer directions; (ii) multivariate dataset with uploads only; (iii) multivariate dataset with downloads only.

4.6.2 Temporal Prediction Model

Based on the dataset analysis considerations presented in Section 4.3.2, we develop and evaluate multivariate RNN models (both LSTM and GRU-based deep learning models) to forecast GridFTP transfers. Our multivariate models can make single-step forecasts (predicting a single future transfer) or multi-step forecasts (predicting a sequence of future transfers). We build separate models for each dataset described in Section 4.6.1. As before, our multivariate predictive analytics system chooses the best model on a per-dataset basis to make useful forecasts. For the single-step forecast case, to predict the future value $G_{k,t+1} = g_{(a_1,\cdots,a_5),k,t+1}$, we rely on past T observations, i.e. $\sum_{\tau=t-T}^{t} G_{k,\tau} = \sum_{\tau=t-T}^{t} g_{(a_1,\cdots,a_5),k,t}$, where $(a_1, \cdots, a_5)$ combined with t forms the 5-tuple. For the multi-step forecast case, we rely on past T observations, i.e. $\sum_{\tau=t-T}^{t} G_{k,\tau} = \sum_{\tau=t-T}^{t} g_{(a_1,\cdots,a_5),k,t}$ to predict a sequence of N future values, $\sum_{n=1}^{N} G_{k,t+n} = \sum_{n=1}^{N} g_{(a_1,\cdots,a_5),k,t+n}$. A similar strategy is used for the directional transfer datasets. We use a 3-layer deep RNN with 64, 32 and 16 GRU cells for multivariate forecasting. A dropout layer is added to final hidden layer with a probability of 50% to avoid overfitting. We also note that Adam optimizer [105] was used with a training batch size of 64, with a linear activation

function at the output layer. Additional parameters of our multivariate model is detailed in Table 4.1. With the exception of additional data processing for multi-step multivariate inputs, the RNN design remains the same.

4.6.3 Performance Evaluation

We compare our deep RNN models with a multi-layer perceptron (MLP) deep learning model. Our multi-layer perceptron (MLP) predictor consisting of three dense hidden layer with 64, 32, and 16 fully connected units, a dropout layer added to the final hidden layer with a probability of 50%, hyperbolic tangent activation function at the hidden layers and linear activation at the output layer. We compare the performance of our model with the MLP predictor using three widely used performance metrics namely: (i) Mean Absolute Error (MAE), (ii) Mean Squared Error (MSE), and (iii) the coefficient of determination (r^2 score). We have also presented the root mean squared error (RMSE) for convenience. The dataset used in making the predictions, $\mathbf{G}_m$ (1-minute aggregate granularity), was measured over 7 days and contains 1,004,300 GridFTP transfer connection records from six user roles described in Section 4.3.1. This dataset was partitioned into a training set and a validation set in the ratio of 80% and 20%, categorized by user roles. We also evaluate two types of input scaling, namely min-max scaling (See Equation 4.1) and standard scaling. Standard scaling on a dataset $\mathbf{G}$ is defined as:

$$\hat{\mathbf{G}} = \frac{\mathbf{G} - \mathbf{G}_\mu}{\mathbf{G}_\sigma} \tag{4.15}$$

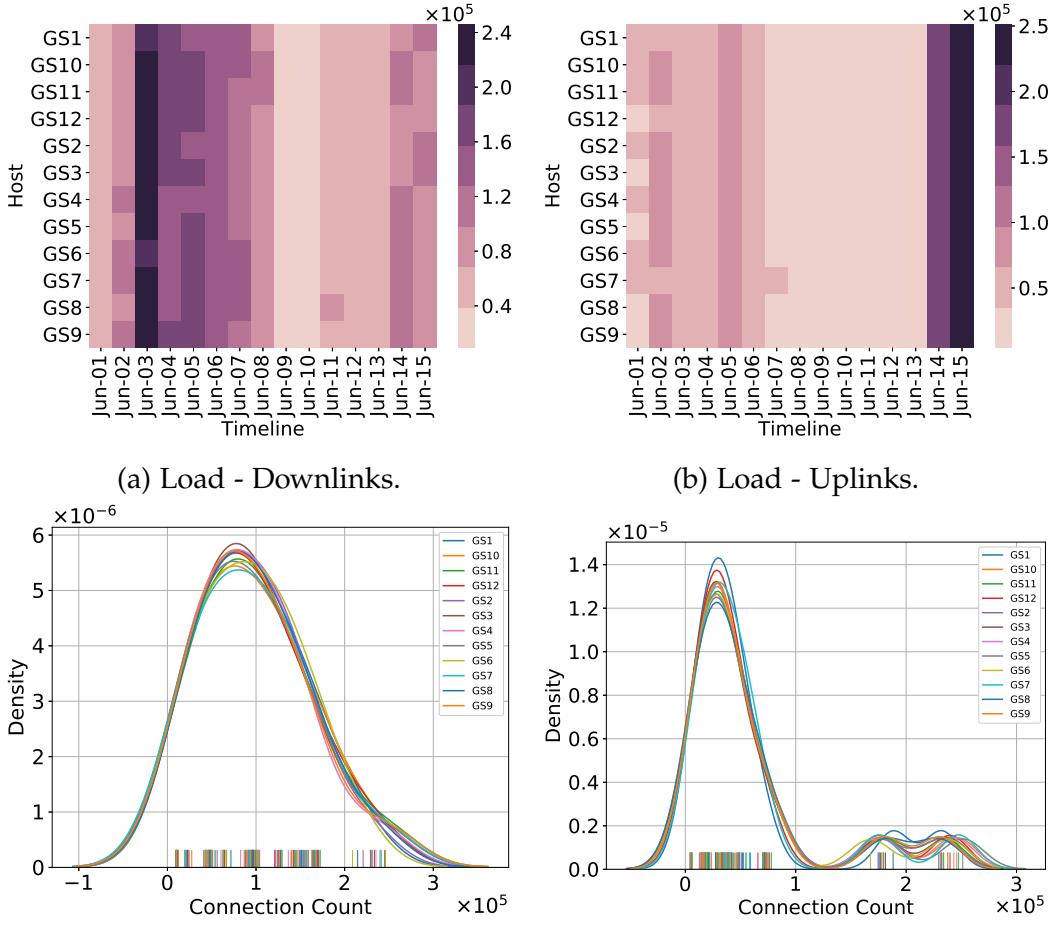
where $\mathbf{G}_\mu$ and $\mathbf{G}_\sigma$ represents the dataset mean and standard deviations, respectively. Next, we present our deep RNN models' prediction results and compare them with the MLP forecasting model.

4.6.4 Prediction Results and Discussion

We present the prediction performance results for the multivariate datasets $\mathbf{G}_m$ (combined upload and download connections), $\mathbf{D}_m$ (downloads only) and $\mathbf{U}_m$ (uploads only), each at one-minute aggregate granularity. The models' prediction performance with min-max scaling and standard scaling are presented in Figures 4.9 and 4.10, respectively. In both cases, we present the MAE, MSE, RMSE and r^2 scores. Our proposed models, depending on the user role, shows an improved error performance between 19.94%–32.56%, 14.02%–29.52%, and 7.35%–16.05% regarding MAE, MSE and RMSE, respectively over the MLP model with min-max input scaling. Our models also show r^2 score improvements between 20.76%–66.49% over the MLP model with min-max input scaling. Further, for the standard input scaling case, our models, in comparison to the MLP model, shows an improved error performance between 6.65%–13.85%, 3.86%–6.77%, and 1.95%–3.44% regarding MAE, MSE and RMSE, respectively. It also shows r^2 score improvements between 17.88%–35.94% over the MLP model. The above results show the effectiveness of our multivariate LSTM and GRU-based deep RNN models for GridFTP load forecasting. When comparing the multivariate RNN models, we observe that the LSTM and GRU models' performance is within 1.5% of each other for the above performance metrics. In the next section, we demonstrate how accurate per user role predictions can be effectively used to develop intelligent load balancing schemes for the LVS [98] cluster.

4.7 Application-aware Load Balancing

Data-intensive science applications, with users interacting with massive amounts of data, place dynamically varying demands on the network infrastructure. However,



(a) Load - Downlinks.

(b) Load - Uplinks.

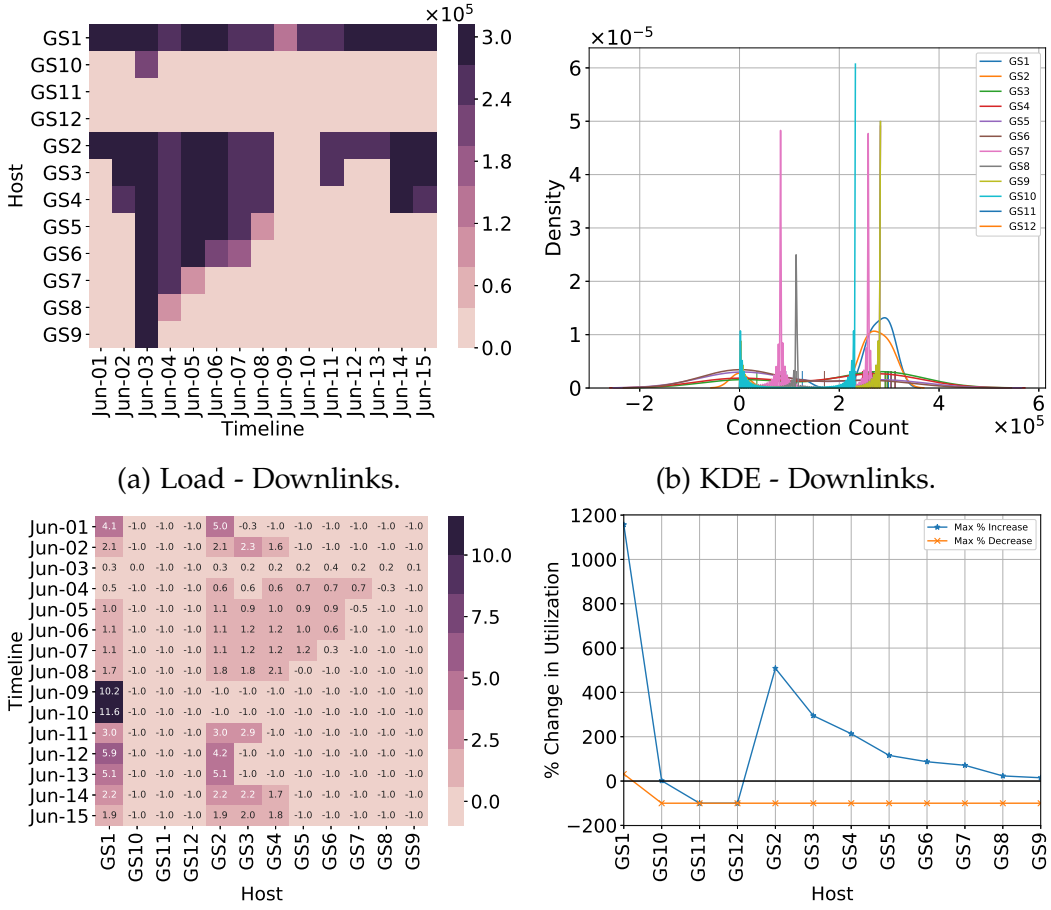
(c) KDE - Downlinks.

(d) KDE - Uplinks.

Figure 4.11: LVS weighted least-connection scheduling load distribution (15 Days).

conventional campus network and supercomputing center architectures, without a global view of the network, rely on load balancers that are not precise. With the emergence of SDN, significant research has gone into developing accurate load balancing methods with better performance than their conventional alternatives [88]. However, limited work has been done in developing efficient load balancers capable of handling massive amounts of data transfers intelligently from high-throughput distributed computing workflows.

The Linux Virtual Server (LVS) [98] is a high-availability, highly-scalable load



(a) Load - Downlinks.

(b) KDE - Downlinks.

(c) Daily % change/server.

(d) Change in utilization/server.

Figure 4.12: APRIL scheduling load distribution (15 Days).

balancing solution built on a cluster of real servers. The LVS architecture is fully transparent to end-users/applications and behaves as a single high-performance virtual server. LVS is a widely used open source load balancing solution in many supercomputing centers. LVS implements several load balancing schedulers including (weighted) round-robin, (weighted) least-connections, source/destination hashing, and locality-based least-connection schedulers. While these schedulers perform adequately, they do not provide fine-grained controls for intelligently balancing connection loads based on application behavior.

The use of application-layer metadata benefits load balancing systems by allow-

ing them to make intelligent decisions based on application behavior. However, such application metadata exchange is often *limited or nonexistent*. In the following, we propose an intelligent load balancer that exploits both application-awareness and predictive analytics knowledge to provide fine-grained load balancing controls.

4.7.1 Application-aware Predictive Intelligent Load Balancer (APRIL)

We propose APRIL, an application-aware, predictive, intelligent load balancer. APRIL intelligently combines application-layer metadata with deep learning predictive analytics to create customized load balancing policies. Our proposed approach is highly adaptable to both end-user/application requirements and behavior while providing fine-grained controls to the site administrator to prioritize or isolate desired flows.

We demonstrate an approach that exploits application-awareness and per user role forecast information to maximize GridFTP server utilization. The proposed approach, APRIL is described in Algorithm 4.1. First, we define per-server maximum capacity and weights. The weights are used to decide the preferred order of load distribution among the servers upper-bounded by their capacity. The weights are (re)adjusted based on the per user role forecast information periodically to ensure that the utilization is maximized. We formally define the problem as:

$$Min. \left(\frac{C_n \cdot S_k}{W_k} \right), \forall n \in N, \forall k \in K \tag{4.16}$$

where, C_n represents the number of connections across K servers S_k, each weighted by W_k. The weight updates, W_k for each server S_k is defined as:

$$W_k = \alpha_\kappa \kappa_{S_k} \times W_{k-1} \times \frac{|C_{act}|}{|C_{pred}|}, \forall k \in K, \forall \alpha_\kappa \in (0, 1] \tag{4.17}$$

Algorithm 4.1 APRIL($\mathbf{G}, S_k, \alpha_k$)

Require: Connection dataset ($\mathbf{G}$), Servers S_k, Capacity threshold α_k.
Output: Load distribution.

$\kappa_S = \alpha_k \cdot \kappa_{S_k}, \forall \alpha_{S_k} \in (0, 1]$
$W_k = 1, \forall s \in S_k$
for $C_{act} \in \mathbf{G}$ **do**
  Compute $C_{pred,t+1} = \text{RNN}(C_{act}, \tau), \forall \tau \in (t - \tau, t)$
  while $C_{act} \neq \phi$ **do**
    Find $s \in S_k$ with the smallest κ_S
    for $s \in S_k$ **do**
      if $\kappa_S \leq \alpha_k \cdot \kappa_{S_k}$ **then**
        $s := \{C_{act,t}, W_k\}$
        $W_k := \alpha_\kappa \kappa_{S_k} \times W_{k-1} \times \frac{|C_{act}|}{|C_{pred}|}, \forall k \in K$
        break
      else
        Find $s \mid \kappa_S \leq \alpha_k \cdot \kappa_{S_k}$
      end if
    end for
    $W_{k-1} := W_k$
    $C_{act} := \{C_{act} \cup C_{pred}\}$
  end while
end for

where, κ_{S_k} is the current server capacity; α_κ is the capacity threshold. W_k and W_{k-1} are the current and previous weights, respectively. C_{act} and C_{pred} represent the total current and predicted connections, respectively. Each servers' weights are adjusted based on the predictions for that observation period. By using application-layer metadata and per user role forecast information, we can maximize the server utilization by assigning the appropriate weights for each server. The weights also ensure that an appropriate number of connections live on each server without exceeding the capacity (*viz.* controlled by α_κ).

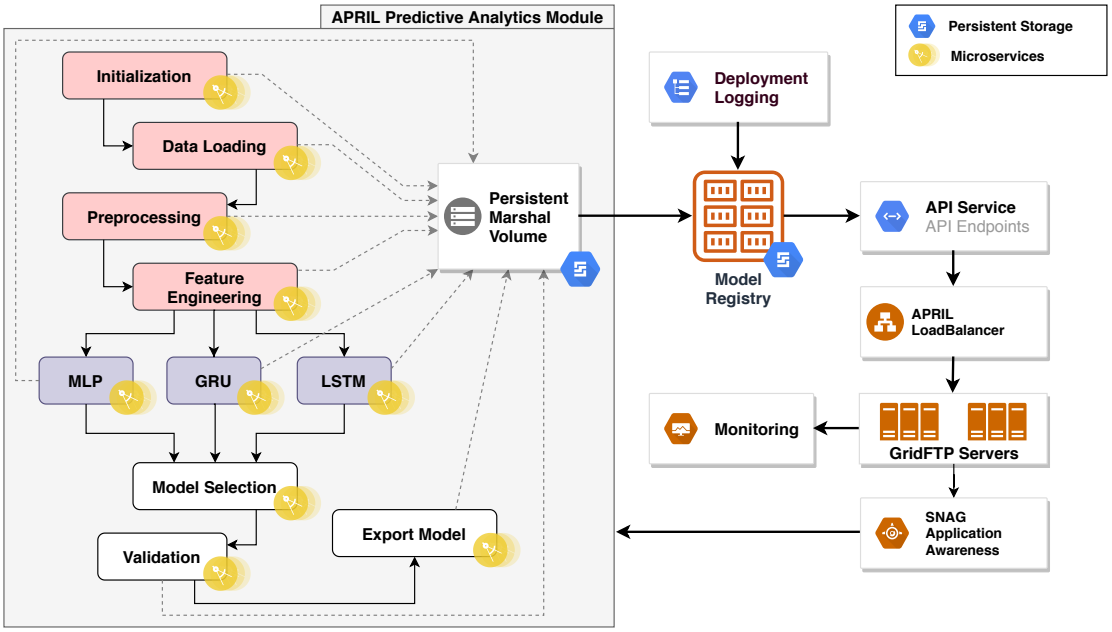


Figure 4.13: APRIL Deployment.

4.7.2 Results and Discussion

First, we present our experiences with the LVS weighted least-connection (WLC) scheduling, which is the primary scheduler used in our production U.S. CMS Tier-2 site. Figure 4.11 shows the LVS WLC scheduling heatmaps for the 12 GridFTP servers (labeled *GS1–GS12*) in the production network. The Figures 4.11a and 4.11b show the downlink and uplink connection distribution, respectively, when LVS WLC is used. The corresponding kernel density estimates (KDE) shown in Figures 4.11c and 4.11c indicate that load is almost equally distributed across all servers. The distribution performance of our proposed method, APRIL, is shown in Figure 4.12. From the heatmap shown in Figure 4.12a, we see that APRIL is better at redistributing loads with an objective of maximizing server utilization. This is also confirmed by the KDE in Figure 4.12b, which shows the difference in probability density for servers with increased utilization. Lastly, we show the resulting daily percentage change effected by APRIL in each server when compared

to LVS WLC, in Figure 4.12c. We observe that our approach simultaneously maximizes utilization (up to 11 times increase) in some servers while reducing utilization significantly in others (a minimum of 0.54 times decrease). The per-server (maximum and minimum) percentage change averaged over 15 days is also presented in Figure 4.12d.

Although we have mainly presented the results by comparing our approach with LVS WLC, we note that other LVS scheduling algorithms were also evaluated during our experiments. Specifically, we configured LVS to use three additional schedulers on the production network namely: round-robin (both pure and weighted), source hashing, and destination hashing. Other than WLC, these other schedulers exhibited unstable behavior for opportunistic transfers such as LIGO workflows. This resulted in frequent dropped connections in the production network and server loading problems, and therefore we had to revert to WLC for stable network operation. APRIL provides the ability to accurately forecast the underlying network flows. Our approach, when combined with network functions virtualization (NFV) techniques such as service function chaining (SFC) allows site operators to conveniently impose various network policies. These approaches improve the end-to-end GridFTP transfer performance through multi-path bulk data movement by employing XSP, GridFTP striping and data-path parallelism. This allows for optimizations in resource allocation, site-specific policy enforcement, resource-usage accounting and traffic prioritization.

4.8 APRIL Deployment Strategy

In this section, we discuss the APRIL deployment strategy and integration with the GridFTP servers. The proposed APRIL deployment strategy is shown in

Figure 4.13. The deployment process consists of three components, namely:

1. APRIL predictive analytics module
2. Model registry and associated service API endpoints
3. APRIL load balancer and GridFTP integration

4.8.1 APRIL Predictive Analytics Module

The APRIL predictive analytics module consists of machine learning workflows and includes various tasks associated with data processing, model training and validation. We partition the machine learning workflows into initialization, data acquisition, data preprocessing, feature engineering, model training, model selection, model validation tasks. Numerous deployment strategies are available for training and serving models in production, such as batch prediction and real-time prediction across various service options such as web services, serverless functions (e.g., AWS Lambda, Google Kubernetes Engine), containers and notebooks. To ensure deployment scalability, portability and modularity, we design each of the above workflow tasks as microservices. The resulting pipeline employs a persistent storage volume for marshaling the model artifacts. The pipeline is represented by a directed acyclic graph (DAG) where each workflow task is related to other tasks in the pipeline through one or more dependencies. Each task in the pipeline runs a single job and affects other workflow tasks down the pipeline. Further, each workflow task in the pipeline is a self-contained unit that provides services to other tasks in the workflow. The microservice architecture ensures that workflow tasks can be easily updated, replaced and monitored.

The APRIL predictive analytics module pipeline is shown in Figure 4.13. The initialization task loads the necessary libraries, environment variables, storage

paths and initializes the persistent marshal volume for model state management. The data acquisition task implements a REST API to fetch GridFTP connection and transfer information from the SNAG [16] application-aware module. SNAG exposes APIs to allow access to the GridFTP application-layer metadata. This data is sanitized, preprocessed and fed to a feature engineering task by the data preprocessor. The feature engineering task ensures that the right features are selected and appropriately scaled for use by both univariate and multivariate models. Next, univariate and multivariate RNN and MLP models are separately trained, and the best models are selected. We evaluate the selected models against a validation set, and the models are exported and stored for deployments.

4.8.2 Model Registry and API Services

The models exported from the machine learning workflow pipelines are stored and managed in a model registry. The model registry is responsible for the lifecycle of the trained models and provides a convenient service to manage and track the deployed model artifacts. The models are then exposed using web service APIs (developed in Python Flask). The APRIL load balancing system utilizes the models' prediction service API endpoint to create load balancing strategies. A single API service endpoint is provided for each trained model. We use Continuous deployment (CD) to automate model deployment and service exposure when newly trained models are available. The CD pipeline is set up with our API service to load the updated model artifacts and manage the API entry point functions. We monitor the model registry and the API service performance using deployment logging and instrumented services. Our instrumented services expose Prometheus metrics that are aggregated and monitored by the SNAG Elastic cluster monitoring system (See Section 4.4.3). However, we note that the instrumented services can

also be monitored separately using systems such as Prometheus and Grafana. We note that the APRIL machine learning pipeline workflows and the model registry use separate persistent storage volumes. Next, we present the APRIL load balancer integration with the GridFTP servers for application-aware and intelligent load balancing.

4.8.3 APRIL Load Balancer and GridFTP Integration

The SNAG application-aware module updates the Elastic cluster with information about all current and ongoing GridFTP data transfers in real-time. The APRIL data acquisition system updates its datasets with the new time series, periodically, resulting in new RNN models and an updated prediction service. The APRIL load balancer periodically queries the prediction service APIs to obtain per-user role GridFTP connection forecasts. The forecast information is used to create load distribution policies as described in Algorithm 4.1. These policies are used to adjust the GridFTP servers' load distribution strategies. The GridFTP server loads and performance are monitored to ensure smooth operation. The APRIL predictive analytics module, the model registry and its APIs, the load balancing system and the GridFTP servers with the SNAG application-awareness form a closed loop system. Thus, APRIL continuously improves GridFTP server utilization through application-aware and intelligent load balancing.

4.9 Conclusions

We proposed an application-aware intelligent load balancing system (APRIL) for high-throughput data-intensive science workflows such as CMS and LIGO. Our proposed solutions used a real dataset representing *800 million* GridFTP

transfer connections from a major U.S. CMS Tier-2 site. We presented an extensive analysis of this dataset to identify long-term temporal dependencies between different user roles and workflow memberships. Using the insights from the data analysis, we leveraged deep learning techniques for time-series modeling to develop an application-aware predictive analytics system using long short-term memory (LSTM) and gated recurrent units (GRU) based recurrent neural network (RNN). Our deep RNN predictive analytics system accurately forecasts GridFTP connection loads and performs better than ARIMA or multi-layer perceptron (MLP) models. We then developed a novel application-aware, predictive and intelligent load balancer, APRIL, that effectively integrates application metadata and load forecast information to maximize server utilization. Through extensive experiments, we demonstrated the effectiveness of APRIL by comparing it with an existing production Linux Virtual Server (LVS) cluster. Our approach improves server utilization, on an average, between 0.5–11 times over its LVS counterpart. Lastly, we presented a deployment strategy for integrating APRIL with the GridFTP ecosystem. The deployment includes scalable and modular components, including workflow task separation using the microservices architecture, model registry and an extensible services API. This approach allows for easy maintenance and monitoring of the APRIL system. Our future work will focus on developing load balancing schemes that will consider a broader range of application metadata parameters.

Chapter 5

Scalable Application-aware Edge and Generalized Service Performance Measures for Multi-Cluster Distributed Service Mesh Architectures

5.1 Introduction

The evolution in cloud computing [107] has led to a fundamental shift in application development and service delivery paradigms. To ensure rapid service instantiation, service agility and flexible deployments in distributed cloud infrastructures, operators increasingly rely on cloud-native applications [108] and microservice architectures [109, 110]. To fully realize the benefits of cloud-native systems, automated microservice deployment and orchestration systems are necessary (e.g., [111, 112]). Cloud-native architectures disaggregate applications' control and data planes, thereby ensuring flexibility, programmability and resilience. A key challenge in this approach is reliable service-to-service communication across distributed infrastructures.

The network service mesh (NSM), or "the service mesh," is an emerging pattern that enables inter-service communication by providing an addressable infrastructure layer. Service meshes provide declarative control over network behavior

through policy-based configuration, management and monitoring frameworks. Typically, service meshes offer uniform observability, transparent service traffic visibility, granular traffic control, resilience features such as circuit-breaking, policy-based security and eventually consistent service discovery. While container orchestrators focus on workload management, scheduling, health monitoring and discovery, a service mesh focuses on mitigating unmet service-level requirements. Service meshes ensure secure service-to-service communication and service discovery across distributed infrastructures using a network of sidecar containers or proxies. These sidecars/proxies form the service mesh data plane and are responsible for relaying traffic securely between services and client endpoints. Further, they are combined with ingress (reverse proxy) and egress (forward proxy) gateways to control inbound and outbound traffic, respectively.

With the emergence of the Internet of Things (IoT) in domains such as smart homes [113], smart cities [114, 115], health care [116], agriculture [117, 118], etc., there is an increasing push towards integrating heterogeneous devices, machines and digital objects with automation in the virtual world. Integration solutions for IoT systems frequently communicate with infrastructures in the edge/cloud for data-analytics, intelligent decision-making, and automation. IoT and data analytics enable smart agriculture to achieve better operational efficiencies and improve crop yields. Smart and precision agriculture [119] relies on sensor network deployments for various tasks, including crop health monitoring, process control and automation. IoT integrates numerous existing technologies, including (wireless) sensor networks, access technologies, connectivity solutions, computing and end-user applications. Although IoT empowers agriculture with smart and intelligent decision-making tools to integrate agricultural implements, knowledge and services for improved productivity and yield gains, critical challenges exist.

In contrast to conventional wireless sensor network deployments [120], IoT in precision agriculture requires advanced capabilities and tight integration between various technologies such as in-field connectivity, communication devices, aerial or satellite imaging systems, real-time control systems, data analytics and computing systems to be effective.

In-field sensors and imaging systems generate large quantities of high-resolution datasets. Scalable and timely processing of in-field data can provide valuable insights for the specific needs of farmers, crops and soil. However, real-time data processing to obtain such insights can be challenging due to the lack of high-performance computing infrastructure, locally. Numerous solutions propose using cloud resources for data processing and decision-making insights, and they assume the availability of adequate bandwidth and connectivity for high-speed data movement. Although a high-bandwidth, wide-area network can remedy the weak or nonexistent coverage in rural areas, it is typically unavailable. Further, the lack of standardization in how devices and most sensors communicate, both with each other and with external data acquisition/processing systems, creates heterogeneity and exacerbates the above problem. Thus, a key challenge is to provide a *scalable, low-cost edge computing solution for environments (e.g., rural areas) characterized by limited/intermittent connectivity* that are typical of agricultural Internet of things (Ag-IoT) ecosystems.

Numerous research efforts have also focused on the inter-service communication for networked services, including service composition, service monitoring, (client-side) load balancing and traffic management. Traffic management methods for inter-service communication in the literature rely on network proxies, specialized packet encapsulation (e.g., network service header (NSH) [121]), or overlay networks (e.g., Open vSwitch [122], Tungsten Fabric [123]). Solutions requiring

specific packet headers or overlay networks add complexity to application and service development as they require modified packet processing libraries or full overlay network implementations. While service proxies or sidecars typically used in modern service meshes have the benefits of being platform-agnostic, they result in increased resource utilization due to the injection of sidecar containers for each application or service workload. Therefore, inter-service communication (intra- or inter-cluster) must be optimized across the service mesh as it is critical to ensure predictable service performance. This requirement motivates the need for a global, metric-centric measure of service performance between any service-pair in the service mesh network.

First, we propose ERGO, a scalable edge computing architecture for Ag-IoT environments. To motivate the problem, we present an exemplary image processing application describing the computational/service requirements of an edge computing solution in Ag-IoT settings. ERGO works in connectivity-challenged environments, with limited (possibly periodic) or no wide area network (WAN) connectivity. Further, due to limited hardware resources available to the edge computing infrastructure, we design ERGO to be highly composable and scalable to handle the dynamic needs of Ag-IoT devices. Next, we present the service infrastructure of our system and develop representational state transfer (REST) application programming interfaces (APIs) that IoT devices use to interact with ERGO for various tasks (both management and end-user), including data transfers, decision-insights, service configuration and management. We also present extensive performance evaluation of our proposed solution. Our findings motivate the need for developing edge computing systems that can operate efficiently, independently of cloud-backed assistance for extended durations to serve Ag-IoT needs.

Next, we propose a novel generalized service performance measure (GSPM) to provide a metric-centric view of connected services in a network service mesh. We build a service mesh network by interconnecting multiple ERGO-like clusters to develop reliable service-to-service communication solutions across distributed infrastructures. First, we present a service mesh reference architecture and describe the inter-cluster service-to-service communication operation. We also present a comprehensive model for a two-dimensional (2D) network service mesh. We model two service mesh structures: a simple 2D service mesh and an augmented 2D service mesh. Next, we present the average service distance (ASD) measure and analyze its performance for both service mesh structures. While the average service distance forms a fundamental measure for understanding service performance, distributed services in real-world deployments often employ performance metrics that exhibit large variability across clusters. To account for a metric-centric view of service performance, we develop a generalized service performance measure (GSPM) by augmenting the average service distance with localized metrics that are representative of the service/workload performance within a given cluster. To the best of our knowledge, this is the first effort to develop a metric-centric service performance measure for connected services in a network service mesh. As the service/workloads' performance metrics provide insight and ensure observability locally, the GSPM is vital to quantify and evaluate the acceptable end-to-end service performance bounds. Lastly, we propose a GSPM-based dynamic request routing solution that uses response latency as the metric; we demonstrate its effectiveness by evaluating it on a real-world service mesh testbed comprising nine clusters.

5.1.1 Contributions and Organization

The main contributions of this chapter are as follows:

1. *ERGO Edge Computing Architecture*: We present ERGO, scalable edge computing architecture for Ag-IoT environments. ERGO provides a RESTful service infrastructure for both end-user and management tasks. We also present an extensive performance evaluation of our proposed ERGO system.

2. *Average Service Distance Measure (ASD)*: We present comprehensive network service mesh models, their analysis, and develop the average service distance measure for two dominant service mesh structures, namely a simple 2D service mesh and an augmented service mesh.

3. *Generalized Service Performance Measure*: We develop a generalized service performance measure (GSPM) to provide a metric-centric view of service mesh performance. GSPM relies on localized metrics representative of the service/workload performance to quantify/evaluate acceptable end-to-end service performance bounds.

4. *Real-world Implementation and Evaluation*: We deploy a prototype service mesh comprising of 9 clusters and demonstrate the effectiveness of GSPM-based dynamic routing for optimum service placement based on desired performance metrics.

The chapter is structured as follows: In Section 5.2, we present an overview of the state-of-art in service mesh architectures, edge computing and the related work; In Section 5.3, we present the ERGO architecture, operations/service framework, application instrumentation, autoscaling, and APIs; In Section 5.4, we present an exemplary ERGO application deployment; Section 5.5 details the prototype cluster hardware and the experimental setup; We present ERGO performance evaluation results in Section 5.6; In Section 5.7, we present our reference service mesh ar-

chitecture and its components; Section 5.8 presents our average service distance models for two use-cases, namely a simple 2D service mesh and an augmented service mesh; In Section 5.9, we develop a generalized service performance measure (GSPM) to provide a metric-centric view of service performance; Section 5.10 details our prototype real-world 9-cluster service mesh implementation; In Section 5.11, we develop and evaluate a GSPM-based dynamic routing technique; Lastly in Section 5.12, we conclude our work.

5.2 Related Work

Service meshes have found applications in numerous domains, including 5G networks [124], IoT [125], Security [126, 127], load balancing [128], service function chains [129, 130] and monitoring [131]. A brief review of service mesh architectures and architecture guidance are available in [132, 133]. These works focus on developing domain-specific solutions using the service mesh architecture. The authors in [134] proposed a skewness-aware matrix factorization (SMF) method to model pairwise RTTs for inter-service communication. They demonstrate that SMF finds a good balance between low-rank matrix factorization and skewed distributions. However, the work focuses only on monitoring pairwise RTTs for performance. Other approaches focusing on latency predictions using the matrix factorization approach include [135, 136, 137, 138, 139, 140]. Distinct from the above, our work focuses on developing a generalized service performance measure (GSPM) that provides a metric-centric view of service performance.

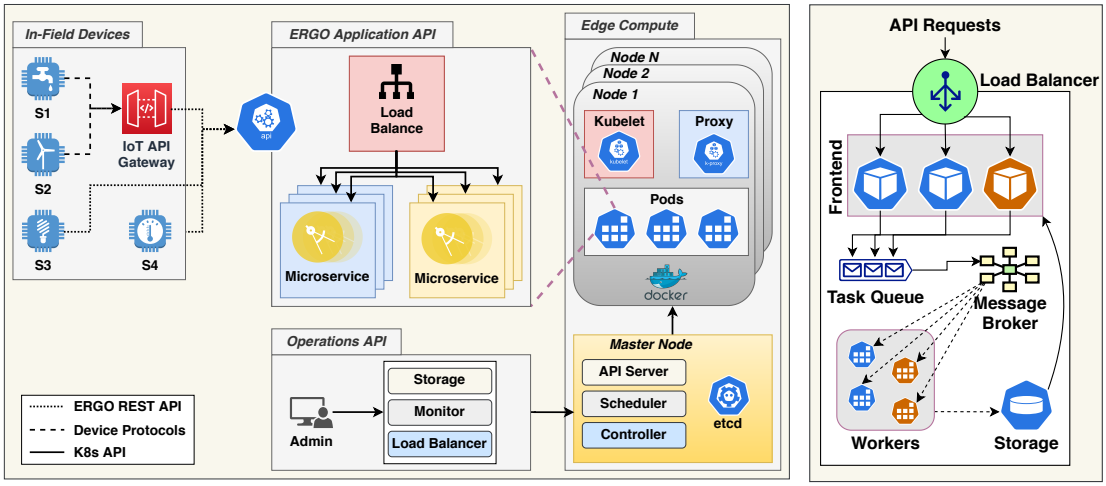
With numerous challenges in agriculture ranging from water scarcity and land degradation due to the excessive use of chemicals, smart farms and precision agriculture have become a go-to solution. Ag-IoT has emerged as a way to

integrate autonomy and precision in this field. Implementing autonomous Ag-IoT systems has not been easy until recent years. However, with the emergence of new technologies and hardware, intelligent Ag-IoT design and implementation are possible. Ag-IoT applications involve several types of data processing, sensor variety, and diverse functionality leading to integrations in text manipulation and image/video processing.

Ag-IoT and Analytics: Elijeh et al. [117] provide an overview of data analytics aspects in Ag-IoT environments. They investigate the main areas that benefit from data analytics in Ag-IoT, namely: (i) prediction, (ii) storage management, (iii) decision making, (iv) farm management (v) precision farming, and (vi) insurance. Numerous works have focused on image processing [141, 142, 143]. Though the application of image processing in Ag-IoT has benefits for automation, deploying such processing-heavy applications is challenging. Any intelligence deployed has to be affordable and close to the farm/actuators to minimize communication delay and costs.

Ag-IoT and Cloud/Edge Computing: Many works have explored the use of a large sensor base for Ag-IoT. These architectures report sensed data to the cloud and employ machine learning [144, 145] for data processing. However, these solutions unreasonably assume broadband coverage allowing farmers to transmit data to the cloud infrastructure [146]. Furthermore, the latencies involved in cloud-based data processing is too large for the real-time data processing needs of Ag-IoT. Many works have explored the integration of edge and cloud [147, 148, 149] to achieve better latency performance. However, they still require Internet connectivity to interact with the cloud. Different from the above, our work focuses on developing an edge-based solution for Ag-IoT environments with WAN connectivity challenges.

5.3 ERGO Architecture



(a) ERGO Architecture. (b) ERGO Microservices.

Figure 5.1: Our ERGO Edge Computing Architecure.

Our proposed ERGO architecture, shown in Figure 5.1 combines container orchestration with scalable web-services to provide Ag-IoT services over RESTful APIs. Our proposed architecture is shown in Figure 5.1a. It shares some common elements with existing edge-cloud solutions such as reconfigurable compute/storage/networking, standardized APIs for application containerization and security features. However, unlike edge-cloud solutions, our architecture is designed to work in resource-constrained environments with limited computational, network bandwidth and power/energy resources. Thus, we focus on seamless operation in a network with heterogeneous protocols and diverse topologies, while addressing the challenges of intermittent/disconnected operation. Our architecture is composed of two essential components, namely: (a) an operations framework and (b) a services framework.

As shown in Figure 5.1, the ERGO operator can manage application/service deployments through the Kubernetes (K8s) [111] operations APIs and can schedule

workloads dynamically. The service framework exposes application APIs to the in-field devices. In-field devices can interact with ERGO either directly or through an IoT API aggregation gateway. Typically, application APIs are exposed through a simple load balancer service. The service framework also provides the ability to compose multiple applications into a single end-user service. The K8s master node runs a *kubelet* instance on each node to configure and manage pod deployments.

5.3.1 ERGO Operations Framework

Our operations framework provides lifecycle management capabilities for various Ag-IoT computing services. ERGO supports both on-demand and scheduled services, in either stateful or stateless configurations. The operations framework relies on an on-premise Kubernetes cluster for container orchestration. All Ag-IoT services available on ERGO are structured as microservices as shown in Figure 5.1b to ensure independently deployable components that are highly scalable. ERGO microservice applications employ Docker as the containerization platform. ERGO reuses the Kubernetes management and operations APIs while accounting for disconnected network operations (see Section 5.3.6). ERGO provides advanced capabilities including (i) layer-2 (L2) load-balancing for both operational and Ag-IoT services, (ii) dynamic storage provisioning service for automated storage lifecycle management of all ERGO stateful applications, and (iii) cluster-wide infrastructure and service monitoring.

5.3.2 ERGO Service Framework

The ERGO service framework exposes services associated with Ag-IoT applications. Both end-users and IoT devices/sensors access these services. We use a cloud-native microservice architecture for all Ag-IoT applications and their associated

services, with each application designed, deployed and managed independently. Microservices communicate with each other through RESTful protocols provided by the container orchestration system. We design all edge microservice applications for reliable operation in disconnected/intermittent internet connectivity scenarios. We implement service APIs using Flask [150] web server gateway interface (WSGI) with a MongoDB [151] datastore for managing and caching applications' stateful information. ERGO microservices are supported on AMD64, ARM64, ARM/v7, and i386 architectures.

5.3.3 ERGO Instrumented Applications

ERGO applications are instrumented and integrated with the clusters' monitoring system. Application instrumentation allows us to profile the application and service performance, thus ensuring reliable network operation. The resulting application/service performance metrics provide runtime intelligence and ensure adequate resource allocation to the ERGO applications. We incorporate numerous performance metrics, including per-endpoint resource usage, request rates, loading information, latencies, and response profiles. A *metrics* end-point (See Table 5.1) is also provided to allow access and integration with external systems. All ERGO instrumented applications integrate with a *Prometheus* monitoring system to expose customized application-level metrics. ERGO queries these metrics in real-time to make autoscaling decisions. Further, we support exporting ERGO application metrics for integration with external data processing and analytic systems through secure *Prometheus* endpoints.

5.3.4 Autoscaling

ERGO supports autoscaling to ensure predictable performance for compute- or network-intensive operations. Our autoscaling algorithm employs pre-defined metrics from instrumented applications to make per-application scaling decisions. The autoscaling system continuously monitors the desired application performance metrics to track user/service loads. It then adjusts the desired number of application microservice replicas automatically to meet performance goals.

We define a set of M performance metrics, $\mu_{(t,i,m)}, \forall m \in M$, measured at time t on instance i. Let the scaling factor $\alpha_{(t,m)}$ represent the number of required application instances at time t based on the performance measure m. Then, the number of instances required to ensure that the performance does not drop below a specified threshold $\tau_{(i,m)}$, for each instance $i = \{1, 2, \cdots, N\}$ and measure $m \in M$, we compute:

$$\alpha_{(t+1,m)} = \left\lceil \alpha_{(t,m)} \frac{\sum_{i=1}^{N} \mu_{(t,i,m)}}{\sum_{i=1}^{N} \tau_{(i,m)}} \right\rceil, \forall m \in M \tag{5.1}$$

In Equation (5.1), $\alpha_{(t+1,m)}$ represents the number of required instances (as measured by m) to ensure predictable performance. The value $\alpha_{(t+1,m)}$ is computed for each metric m, and we set the total number of required instances at time $t+1$ to the maximum of the set as shown in (5.2).

$$\mathcal{A}_{t+1} = \max\{\alpha_{(t+1,m)}\}, \forall m \in M \tag{5.2}$$

Thus, $\mathcal{A}_{t+1}$, represents the total number of instances required to meet the applications' performance needs.

5.3.5 Task Prioritization and Distributed Queues

ERGO supports task parallelism and prioritization through distributed queues. Task queues process heterogeneous datasets aggregated by IoT devices or gateways. We then distribute the aggregated datasets to the appropriate workers using a brokering system (See Figure 5.1b). Using distributed queues and task prioritization capabilities, we ensure the optimal allocation of ERGO computational resources to appropriate tasks. Our task distribution and prioritization system is implemented using Celery, with Redis as the message brokering system. ERGO also features autoscaling based on *task-* and *queue-state* metrics. We can horizontally (# workers) or vertically (resources per worker) autoscale application microservices using a wide range of queue/task based performance metrics, including the number of tasks in the queue, task state (pending, started, running, finished, failed, retry), queue labels or namespaces, task latency, task runtime and current active workers.

5.3.6 Disconnected/Intermittent Connectivity Operations

While the container orchestration platform allows for dynamic management of microservices, disconnected/intermittent connectivity creates new challenges that require special considerations to ensure seamless operation in the event of service failures. ERGO deploys a local image registry and scheduler service to ensure the availability of the application images and data to all nodes in the ERGO edge computing cluster. We also employ an image migration service to update both cluster management and application images periodically.

API	Methods	Description
`/device`	GET, POST, PUT and DELETE	IoT device management and operations APIs.
`/measurement`	GET, POST, PUT and DELETE	APIs for managing device measurements.
`/compute`	GET and POST	APIs for managing various computing functions.
`/metrics`	GET	These APIs provide application performance metrics.

Table 5.1: ERGO Ag-IoT API.

5.3.7 Implementation

ERGO implementation supports various services for device interaction, aggregating measurement information and computing services. Table 5.1 presents an overview of the ERGO APIs and their associated operations. We divide the APIs into the following categories:

1. *Device API*: Manages all IoT/sensor device-related operations such as additions, deletions and updates.

2. *Measurement API*: Handles IoT/sensor device measurements and data aggregating.

3. *Compute API*: Performs the data-specific computational tasks and communicates decisions/outcomes to both end-users and (actuator) devices.

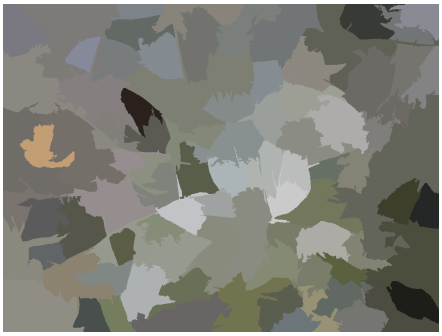
4. *Metrics API*: Application performance metric (APM) end-point for instrumented applications.

We implement our APIs on the Flask WSGI framework. For interactive use, we also expose the API documentation using Swagger. Further, we employ response

marshalling features to format, filter and render expected payload responses. To ensure modularity and to allow for namespace reuse and scalability, we use namespaces to organize the function-specific APIs. We use Flask Blueprints to manage API endpoint prefixes.

(a) Original.

(b) Felzenszwalb.



(c) SLIC.

Figure 5.2: Image Segementation on ERGO.

5.4 ERGO Ag-IoT Application Deployments

Digital agriculture widely uses imaging as a valuable source of information about plant health, terrain utilization, water resources, etc. Commonly employed imaging systems acquire RGB, multispectral and thermal infrared images. These images find applications in 3D mapping, geographic measurements, vegetation detection, health monitoring, irrigation management and thermography. Existing

Figure 5.3: ERGO Prototype Cluster.

cloud-based solutions for agricultural image processing have limited utility in environments characterized by disconnected/intermittent network connectivity. We demonstrate the effectiveness of the ERGO edge computing system through the deployment of an exemplary Ag-IoT application, namely image segmentation and clustering system. The image segmentation/clustering application represents a computationally-intensive task subject to offline processing due to the lack of computing resources locally on the farm.

The ERGO image processing APIs are designed to support various operations such as image transformations, filtering, and segmentation. To demonstrate ERGO APIs for processing Ag-IoT images, we implement two image segmentation techniques that are used by Ag-IoT applications like plant leaf disease detection. Upon receiving an image from the field, we segment it using the Felzenszwalb method and the Simple Linear Iterative Clustering (SLIC) method (See Figure 5.2).

The Felzenszwalb method utilizes a minimum spanning tree (MST) with stopping criteria to prevent the MST from covering the whole image while progressively joining components to create segments. SLIC measures the distance between a pixel and every candidate segment to assign the pixel to a segment. These methods are compute-heavy and provide suitable use-cases for ERGO.

5.5 Experimental Setup

In this section we present the details of the ERGO edge computing cluster hardware components and the datasets used in our evaluations.

5.5.1 ERGO Cluster Hardware

We prototype our ERGO edge computing system on a 5-node Raspberry Pi 4 Model B cluster with quad-core ARM A72 64-bit SoC and 4GB DDR4-3200 SDRAM. The prototype cluster is shown in Figure 5.3. The cluster configuration comprises a single master node controlling 4 worker nodes, connected through an 8-port unmanaged Netgear GS308 GbE switch. We employ an out-of-tree dynamic network file system (NFS) storage provisioner to manage persistent volumes automatically.

Table 5.2: Dataset.

Type	**Parameter**	**Value**
Imaging	Capture Frequency	10 per second
	Image Resolution	1280×960
	Average Image Size	2.5MB
	Spectrum	RGB

5.5.2 Dataset

The imaging dataset is obtained using the autonomous Flex-Ro [152, 153] system. Flex-Ro (shown in Figure 5.4) captures imaging data by navigating the UNL's Field Phenotyping Facility using a differential GPS system. Flex-Ro provides several sensors to measure phenotypic characteristics of plants, including RGB sensors, spectrometers to measure reflectance, temperature/humidity sensors, and ultrasonic sensors to measure height. Flex-Ro captures three images (left, center and right) per measurement, each with a resolution of 1280×960 pixels. Additional dataset details are presented in Table 5.2. ERGO processes the image data using the image processing APIs and provides valuable insights for in-field decisions.

Figure 5.4: Flex-Ro System.

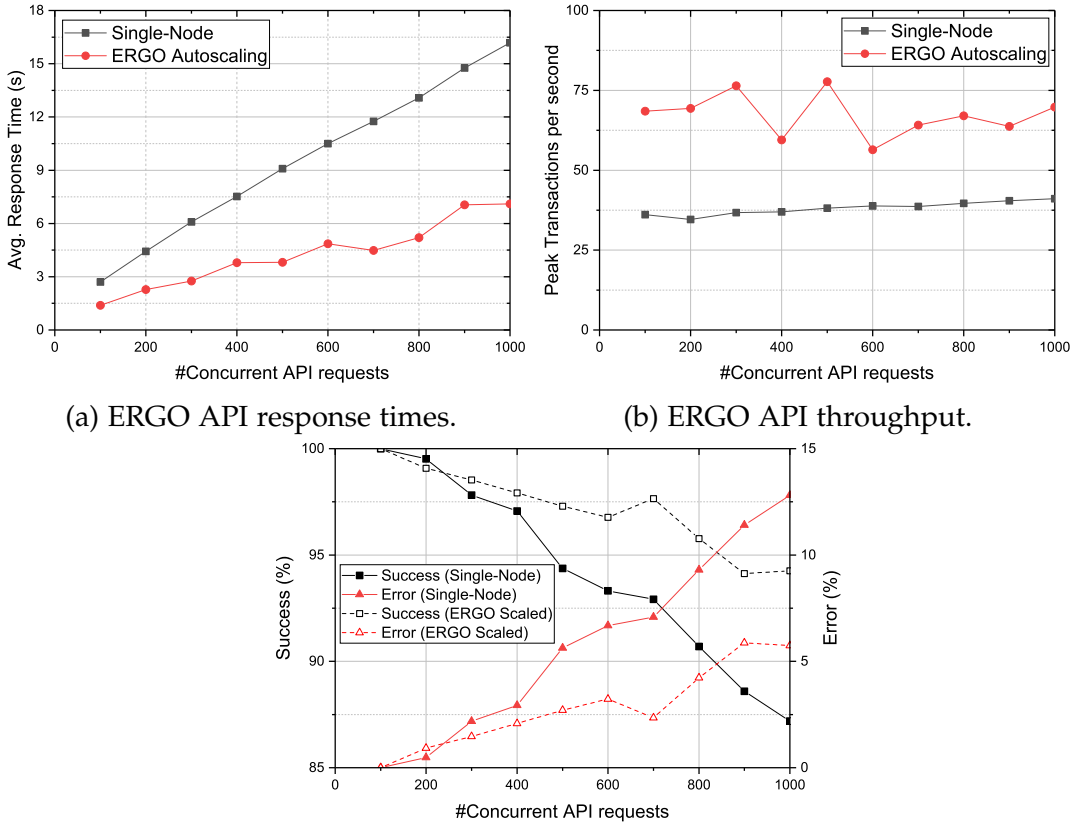
5.6 Results and Discussion

We present the performance evaluation results of our ERGO edge computing cluster in Figures 5.5 and 5.6. We evaluate the performance of ERGO service framework

by load-testing ERGO applications. We also evaluate the ERGO autoscaling system and associated performance benefits including reduced application/service response times and increased peak transaction throughputs. Further, we evaluate ERGO performance and compare it with traditional infrastructure comprising of a single monolithic server (labeled "*single-node*" in the results) with fixed applications. Our evaluation framework consists of an IoT API gateway node interacting with the ERGO cluster wirelessly. The gateway node is built on a Raspberry Pi 3B+ module to aggregate API requests from multiple clients. We employ Apache JMeter [154] for functional load testing and performance evaluation. We present the APIs' response times, throughputs, and connection rates for an increasing number of API requests in Figures 5.5a, 5.5b and 5.5c, respectively.

Our ERGO cluster allocates application/service deployments on 4 worker nodes, each with 4 cores and 4GiB RAM. Our performance evaluations are two-fold. First, we evaluate ERGO performance using a single microservice deployment. Next, we allow the microservice deployment to scale automatically based on resource utilization metrics. We limit the hardware resources to 250 millicores and 512MiB of RAM per microservice. We then evaluate the performance by sending a fixed number of concurrent API requests for 30 seconds. We repeat the test by increasing the number of concurrent API requests to range between 100 to 1,000 per second.

Figures 5.5a−5.5c show ERGO performance with/without scaling. From Figure 5.5a, we observe that ERGO reduces the average response time (between $48\% - 62\%$, and about 54% on average) in comparison to the single-node infrastructure. Further, with autoscaling, we see an increase (between $57\% - 108\%$, and about 77% on average) in the peak transactions per second, leading to increased API throughput as shown in Figure 5.5b. Autoscaling performance with varying



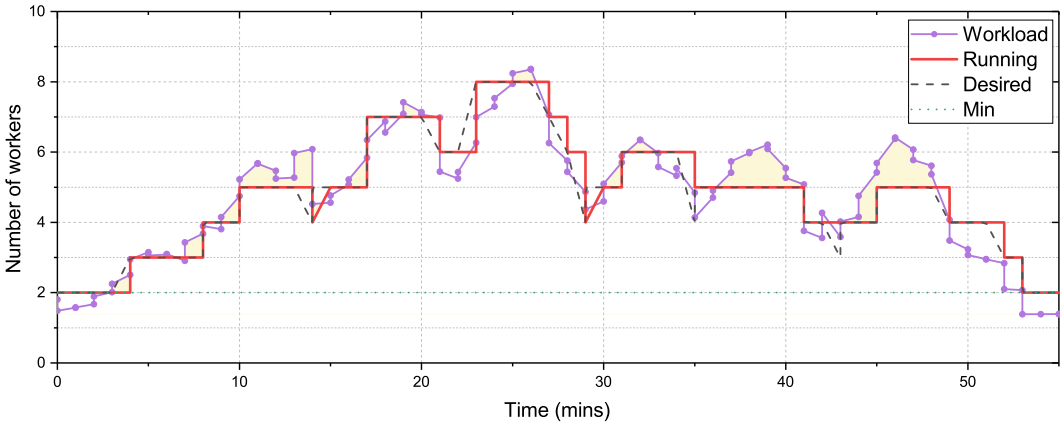
(a) ERGO API response times.

(b) ERGO API throughput.

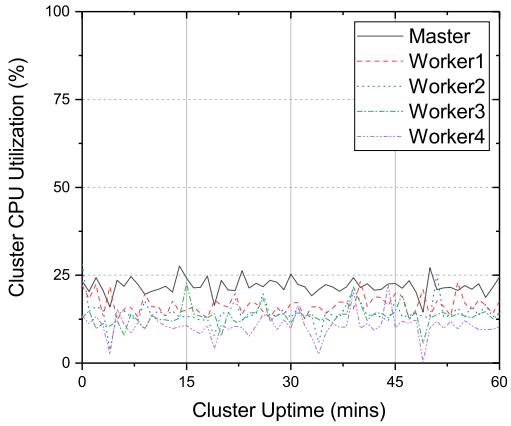
(c) Service success/failure rates.

Figure 5.5: Performance evaluation of the Ag-IoT Application APIs.

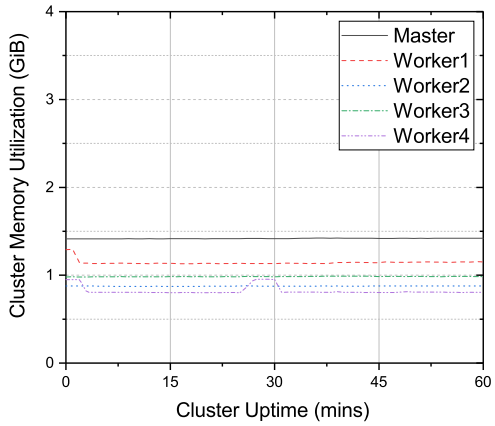
workloads is also shown in Figure 5.6a. Our autoscaler uses CPU utilization and the API request rate as the metrics for up/down-scaling the number of microservice deployments. We set a limit of 8 workers for a microservice and a scale-down delay of 3 minutes. From Figure 5.6a, we see that our autoscaling system reacts well to the changes in the microservices' resource utilization. The shaded area in the figure depicts the events when the service's resource utilization exceeds 80%. We note that the residual capacity of 20% acts as a buffer facilitating seamless operation with the autoscaler spins up new workers. Lastly, we show the ERGO cluster resource utilization (without application workloads) in Figures 5.6b and 5.6c. We note that our 5-node ERGO cluster, with application deployments, utilizes



(a) Autoscaling performance.



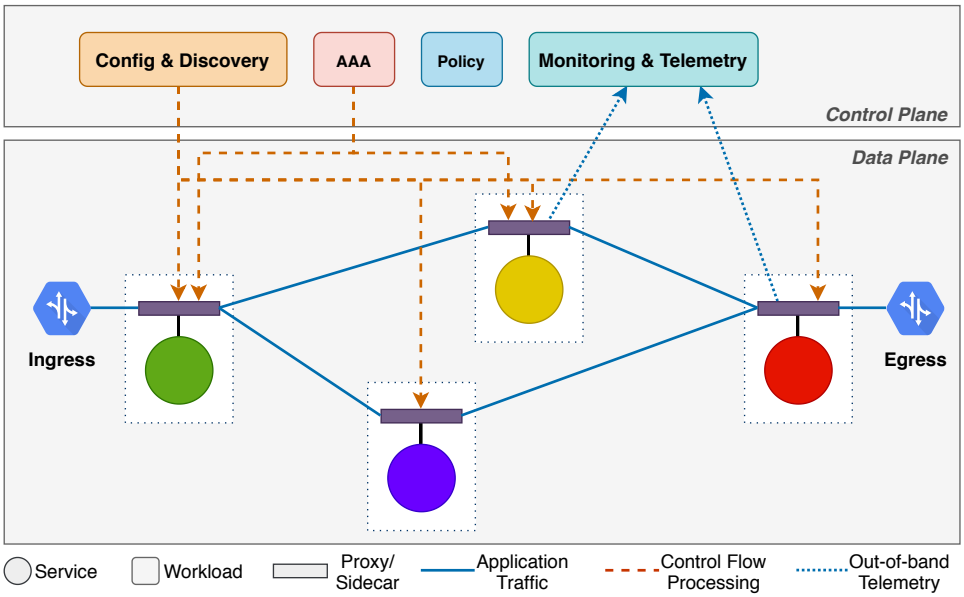
(b) Cluster CPU utilization.



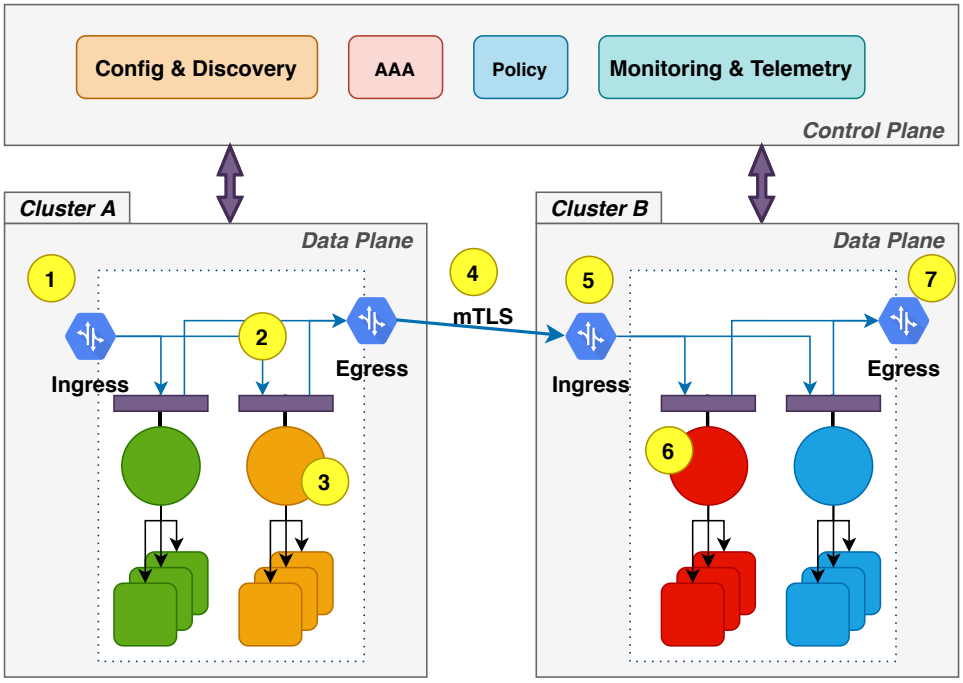
(c) Cluster memory utilization.

Figure 5.6: ERGO autoscaling and cluster performance.

less than 25% of the clusters' hardware resources. This includes load-balancing, dynamic storage provisioning services and cluster monitoring services. Therefore, about 75%-80% of the cluster resources are available for Ag-IoT application or service deployments. Thus, ERGO provides scalable and predictable performance guarantees for an increasing number of services and their processing requirements.



(a) Service Mesh Topology and Operation



(b) Intercluster Service-to-Service Communication

Figure 5.7: Service Mesh Architecture and Components.

5.7 Service Mesh Architecture and Modeling

In this section, we introduce a service mesh architecture, its communication model and present a modeling framework for computing the average service distance (see Section 5.8) for two essential 2D service mesh models, namely a simple service mesh model and an augmented service mesh model.

5.7.1 Service Mesh Architecture

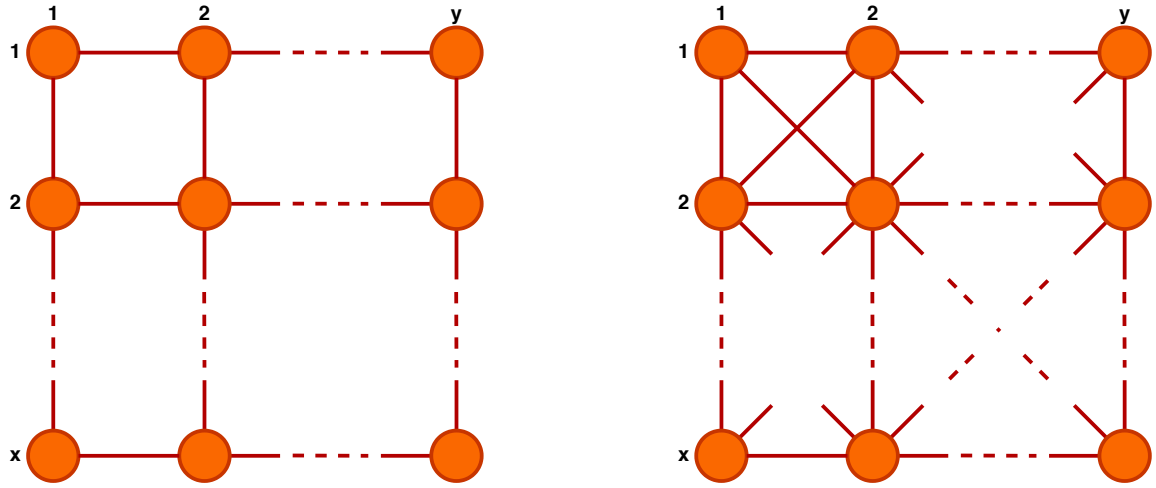
We study a service mesh architecture comprising of two planes: (i) a data plane that performs routing, packet forwarding, load balancing and policy enforcement tasks, and (ii) a control plane tasked with monitoring, network state/topology management, service discovery, identity management, policy and configuration. Figure 5.7 shows the architecture and components of a typical service mesh system. First, we present an exemplary service mesh topology and its operation in Figure 5.7a. Each service mesh is composed of numerous services that are typically separated by trust boundaries (within or across clusters). Service proxies or sidecar containers transparently intercept the service traffic and are responsible for routing, monitoring, authorization, authentication and auditing. The service mesh's data plane manages both inter-cluster and intra-cluster traffic using ingress and egress gateways.

Inter-cluster service-to-service communication operation between two clusters with replicated control planes is shown in Figure 5.7b. The process is as follows: (i) The ingress gateways control inbound traffic into the service mesh. The ingress proxy is akin to a reverse proxy and forwards inbound traffic to the appropriate service. (ii) Service proxies (placed between the service and the application traffic) ensure traffic observability, control and policy enforcement. The proxies handover

the traffic to the services. (iii) The service routes the application traffic to the workloads for processing and the response is forwarded to an egress gateway. (iv) The egress gateway forwards the traffic to the ingress gateway controller of a neighboring cluster. Mutual transport layer security (mTLS) is used to encrypt the traffic between the two clusters. (v) The ingress gateway, like before, forwards the traffic to the appropriate service. (vi) Service proxies ensure uniform observability across the clusters. (vii) The egress gateway sends the response back to the client/application. While we assume that the clusters provide the services, we note that using a similar approach we can integrate services provided by application monoliths, virtual machines (VM) and baremetal workloads.

5.7.2 Service Mesh Models

Typically, service mesh architectures span multiple clusters that are represented by a mesh structure. Due to its topological regularity and network interconnects between cluster pairs, mesh structures are a logical choice for modeling service mesh architectures. While we can measure the global behavior of a service mesh routing algorithm using the service mesh diameter (the greatest distance between any two services in the service mesh) as a metric, its topology dependence, unsurprisingly, results in unrealistic lower bounds for delays during mesh traversals. Specifically, bandwidth limitations, traffic variation, service performance and alternative routes can result in higher delay bounds for a given service mesh architecture. The services' and cluster workloads' dynamic behavior motivates the need for a global quantitative measure of performance between any service-pair in the service mesh network. In the following, we present the average service distance (ASD) as a global measure of the distance between two arbitrary services in a service mesh with a fixed structure.



(a) 2D Service Mesh (b) Augmented Service Mesh

Figure 5.8: Service Mesh Structures.

5.8 Average Service Distance Measure

5.8.1 Preliminaries

To compute the average service distance (ASD) metric, we model two-dimensional mesh structures, as shown in Figure 5.8. We begin with an $x \times y$ service mesh, $x, y \geq 1, \forall x, y \in \mathbb{Z}$. We denote the mesh structure by $M(x, y)$. Each node in the mesh represents a single cluster $c_{i,j}$, composed of a set of services $s_{i,j}^{k}, \forall k \in K$. Each service $s_{i,j}^{k}$ is associated with a set of workloads $w_{i,j}^{k,l}, \forall l \in L$. Without loss of generality, each cluster $c_{i,j}$ is identified by its coordinates (i, j) in the $x \times y$ service mesh network $M(x, y)$, with $1 \leq i \leq x, 1 \leq j \leq y$. Each service $s_{i,j}^{k}$ in $M(x, y)$ generates an arbitrary number of service requests at time t. For modeling, we assume that the service request generation follows a Poisson distribution.

***Definition* 5.1:** *Given a service mesh $M(x, y)$ of size $x \times y$, let c_{i_1,j_1} and c_{i_2,j_2} denote two arbitrary clusters in $M(x, y)$. We define a simple path as a path between two arbitrary clusters traversed only through a set of distinct clusters. We denote by $P_M(c_{i_1,j_1}, c_{i_2,j_2})$*

and $L_M(c_{i_1,j_1}, c_{i_2,j_2})$ the number of such paths between c_{i_1,j_1} and c_{i_2,j_2}, and the sum total of the path lengths in $M(x,y)$, respectively. Then, the average service distance between the clusters c_{i_1,j_1} and c_{i_2,j_2} in $M(x,y)$ is given by:

$$\mathcal{A}_M(c_{i_1,j_1}, c_{i_2,j_2}) = \frac{L_M(c_{i_1,j_1}, c_{i_2,j_2})}{P_M(c_{i_1,j_1}, c_{i_2,j_2})} \tag{5.3}$$

where $L_M(c_{i_1,j_1}, c_{i_2,j_2}) = \sum_{n=0}^{N} n \cdot P_M^n(c_{i_1,j_1}, c_{i_2,j_2})$ for N paths between c_{i_1,j_1} and c_{i_2,j_2}.

Table 5.3: Service Mesh Model Notations.

Parameters	**Description**
$M(x,y)$	A 2D service mesh with $x, y \geq 1, \forall x, y \in \mathbb{Z}$, of size $x \times y$.
$M_s(x,y)$	A simple 2D service mesh with no diagonal links.
$M_a(x,y)$	An augmented 2D $x \times y$ mesh with diagonal links.
$c_{i,j}$	A cluster located at (i,j) in the service mesh $M(x,y)$.
$s_{i,j}^k$	A service in the cluster $c_{i,j}$.
$w_{i,j}^{k,l}$	Workloads associated with the k^{th} service $s_{i,j}^k$ of a cluster $c_{i,j}$.
$P_M(c_{i_1,j_1}, c_{i_2,j_2})$	Total number of simple paths between the cluster pair at (i_1,j_1) and (i_2,j_2).
$L_M(c_{i_1,j_1}, c_{i_2,j_2})$	The sum total of the path lengths.
$\mathcal{A}_M(c_{i_1,j_1}, c_{i_2,j_2})$	The average service distance (ASD) between the clusters at (i_1,j_1) and (i_2,j_2).
$\delta(c_{i_1,j_1}, c_{i_2,j_2})$	The effective distance between the clusters at (i_1,j_1) and (i_2,j_2).
$\mathcal{T}_{M_a}(i,j)$	The transformed average service distance.
$\mathbf{T}(i,j)$	The transformed average service distance transition matrix for the cluster $c_{i,j}$ at (i,j).
$p_{i,j;t}$	A performance metric measured at time t on a service $s_{i,j}^k$ in cluster $c_{i,j} \in M(x,y)$
$\mathbf{P}_M(x,y;t)$	A $x \times y$ matrix with $p_{i,j;t}$ for each cluster in $M_a(x,y), \forall t \in T$.
$\mathbf{G}(c_{i,j};t)$	The generalized service performance measure for $c_{i,j}$ at time t.

5.8.2 Two-dimensional Service Mesh Model

A simple two-dimensional service mesh, denoted by $M_s(x, y)$, where $x, y \geq 1$, is shown in Figure 5.8a. Each cluster $c_{i,j}$ connects to four clusters in this model with the exception of the $2(x + y - 2)$ boundary clusters that connect to two or three clusters. A service mesh routing algorithm will use the l_1 norm (or Manhattan distance) to route a service request between a cluster at (i_1, j_1) to (i_2, j_2).

***Definition* 5.2:** *Let c_{i_1,j_1} and c_{i_2,j_2} represent two clusters located at (i_1, j_1) and (i_2, j_2), respectively. Then, we define the effective distance $\delta(c_{i_1,j_1}, c_{i_2,j_2})$ to be $|i_2 - i_1| + |j_2 - j_1|$. Letting $1 \leq i_1 \leq i_2 \leq x$ and $1 \leq j_1 \leq j_2 \leq y$, we have $\delta(c_{i_1,j_1}, c_{i_2,j_2}) = (i_2 - i_1) + (j_2 - j_1)$.*

From Figure 5.8a, we make the following observations. Consider a service request from from (i_1, j_1) to (i_2, j_2). An efficient service routing approach will reduce the effective distance by a 1 with each traversal on reaching an intermediate cluster. As an example, consider a request from (i_1, j_1) to $(i_2, j_2), 1 \leq i_1 \leq i_2 \leq x, 1 \leq j_1 \leq j_2 \leq y$. With each traversal, an effective routing algorithm will choose links to the right/bottom of the cluster at (i_1, j_1). Further, the chosen paths are also simple and shortest paths. Thus, it follows that the effective path length between $c_{1,1}$ and $c_{i,j}$ is $i + j - 2$.

***Lemma* 5.1:** *Let $M_s(x, y)$ be a simple two-dimensional service mesh of size $x \times y$. Then,*

$$\mathcal{A}_{M_s}(c_{1,1}, c_{x,y}) = x + y - 2 \tag{5.4}$$

While the above analysis assumes that the source of the service request is originating from $c_{1,1}$, Lemma 5.1 holds for any arbitrary source. We can apply a linear coordinate transform to any source to obtain a cluster coordinate pair of

the form in (5.4). While the average service distance for a simple mesh $M_s(x, y)$ is trivial, we present a generalized approach for more sophisticated service mesh topologies by extending a combinatorial process developed in [155].

Let $P_{M_s}(c_{1,1}, c_{x,y})$ and $L_{M_s}(c_{1,1}, c_{x,y})$ denote the total effective paths between the clusters $c_{1,1}$ and $c_{x,y}$, and the sum of those path lengths, respectively. Without loss of generality, as the source is fixed at $(1, 1)$, we represent the parameters by the destination, i.e., (x, y). Using the above analysis (similar to Lemma 5.1), the recurrence relations for $P_{M_s}(x, y)$ immediately follow:

$$P_{M_s}(x,y) = \begin{cases} 1, & x \geq 1, y = 1, \\ 1, & x = 1, y \geq 1 \\ P_{M_s}(x-1,y) & \\ \quad +P_{M_s}(x,y-1), & \text{otherwise} \end{cases} \tag{5.5}$$

When either $x, y = 1$, $M_s(x, y)$ transforms into a one-dimensional array, and therefore we have a single path from the service request source to any destination service. For $x, y \geq 2$, consider the case when the request from $c_{1,1}$ is forwarded to either $c_{1,2}$ or $c_{2,1}$. Using a linear transformation, we observe that the effective path from clusters $c_{1,2}$ to $c_{x,y}$ or $c_{2,1}$ to $c_{x,y}$ is identical to the paths from $c_{1,1}$ to $c_{x,y-1}$ or $c_{1,1}$ to $c_{x-1,y}$, respectively. Further, for each path from $c_{1,2}$ to $c_{x,y}$, the path between $c_{1,1}$ to $c_{1,2}$ contributes a single path to $L_{M_s}(x, y)$. Thus, it follows that the first path contributes $L_{M_s}(x, y-1)$ to $L_{M_s}(x, y)$. The recurrence relation for $L_{M_s}(x, y)$ is therefore given by:

$$L_{M_s}(x,y) = \begin{cases} x-1, & x \geq 1, y = 1, \\ y-1, & x = 1, y \geq 1 \\ L_{M_s}(x-1,y) & \\ \quad +L_{M_s}(x,y-1) + P_{M_s}(x,y), & \text{otherwise} \end{cases} \tag{5.6}$$

Using generating functions [156, 155] to solve the above recurrence relations, we have:

$$\mathcal{A}_{M_s}(x,y) = \frac{L_{M_s}(x,y)}{P_{M_s}(x,y)} = \frac{(x+y-2)\binom{x+y-2}{y-1}}{\binom{x+y-2}{y-1}} \tag{5.7}$$

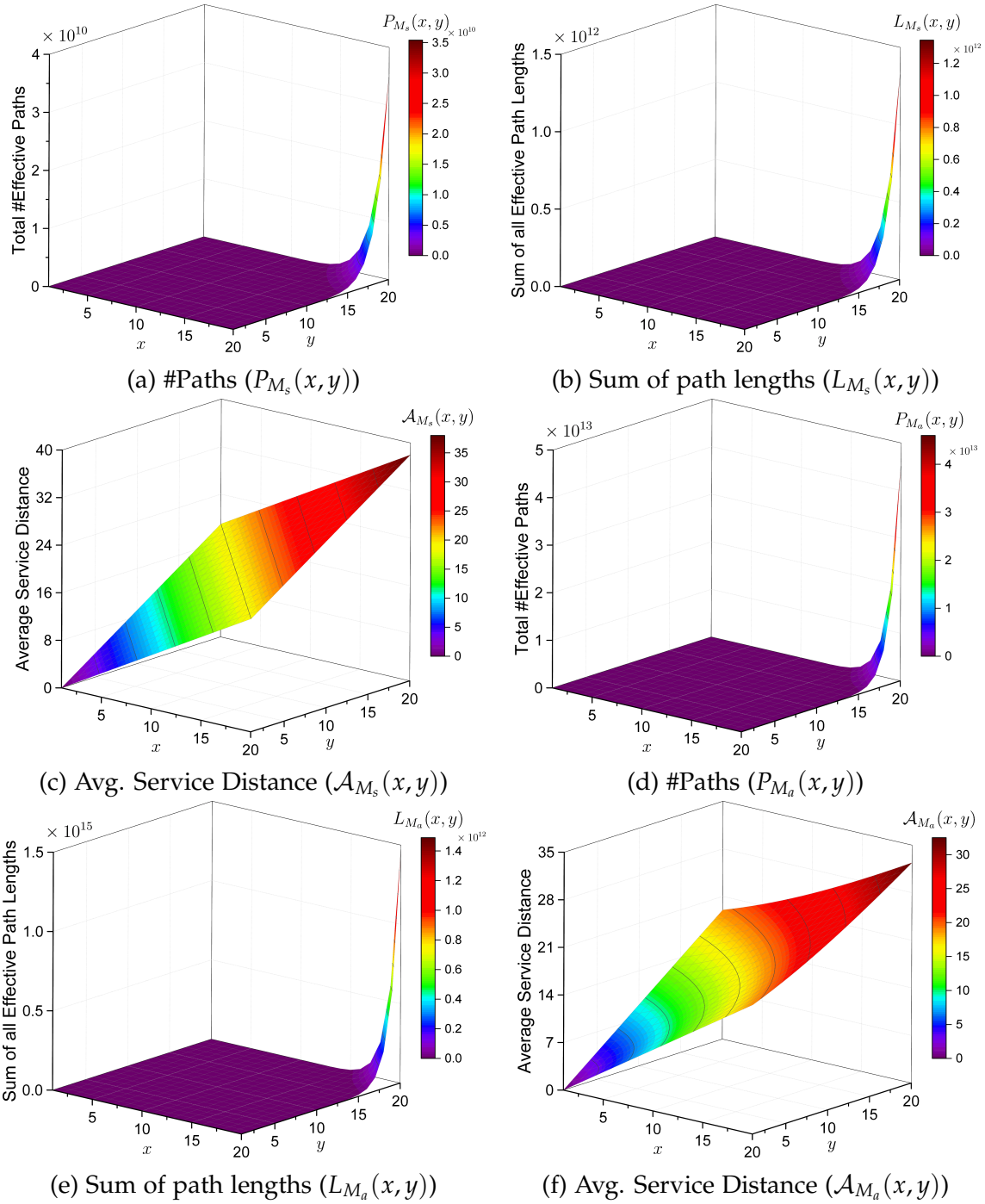
Therefore, the ASD between the clusters $c_{1,1}$ and $c_{x,y}$ is:

$$\mathcal{A}_{M_s}(x,y) = x+y-2 \tag{5.8}$$

Thus, Eqn. (5.8) confirms Lemma 5.1. Figures 5.9a, 5.9b and 5.9c show the variation of $P_{M_s}(x,y)$, $L_{M_s}(x,y)$ and $A_{M_s}(x,y)$ for $1 \leq x,y \leq 20$, respectively. From the figures, we observe that both $P_{M_s}(x,y)$ and $L_{M_s}(x,y)$ grow exponentially with increasing service mesh size.

5.8.3 Augmented Two-dimensional Service Mesh Model

An augmented two-dimensional mesh, denoted by $M_a(x,y)$ is shown in Figure 5.8b. In addition to the paths in $M_s(x,y)$, $M_a(x,y)$ also includes diagonal links. As the *diameter* of the service mesh $M_s(x,y)$ increases linearly with an increase in x and y, the diagonal links in $M_a(x,y)$ serve to reduce the network diameter. These diagonal links consequently lead to a reduction in the average service distance as shown next.



(a) #Paths ($P_{M_s}(x,y)$)

(b) Sum of path lengths ($L_{M_s}(x,y)$)

(c) Avg. Service Distance ($\mathcal{A}_{M_s}(x,y)$)

(d) #Paths ($P_{M_a}(x,y)$)

(e) Sum of path lengths ($L_{M_a}(x,y)$)

(f) Avg. Service Distance ($\mathcal{A}_{M_a}(x,y)$)

Figure 5.9: Simple and augmented service mesh performance.

Consider the effective path between two clusters $c_{1,1}$ and $c_{i,j}$. From Figure 5.8b, it is evident that such a path from $c_{1,1}$ to $c_{x,y}$ will traverse the major diagonal to

a cluster at $(\min\{i,j\}, \min\{i,j\})$ and then reach (i,j) directly. Further, we note that the length of the path is $\max\{i,j\} - 1$. Thus, the augmented service mesh $(M_a(x,y))$ network diameter reduces to $\max\{x,y\} - 1$. When, $x = y$, the number of links in $M_s(x,y)$ and $M_a(x,y)$ are $2x^2 - 2x$ and $4x^2 - 6n + 2$, respectively. In comparison to $M_s(x,y)$, $M_a(x,y)$ has numerous effective paths from $(1,1)$ to (x,y), with the length of the longest path (without diagonal links) equal to $x + y - 2$. Therefore, the effective path length for $M_a(x,y)$ is between $\max\{x,y\} - 1$ and $x + y - 2$.

We denote the total effective paths between the clusters $c_{1,1}$ and $c_{x,y}$, and the sum of those path lengths by $P_{M_a}(c_{1,1}, c_{x,y})$ and $L_{M_a}(c_{1,1}, c_{x,y})$, respectively. While most effective paths between $c_{1,1}$ and $c_{i,j}$ are not the shortest paths, the major diagonal paths connecting the two clusters may be affected by factors such as reduced residual capacity, congestion and bandwidth limitations. Therefore the recurrence relations for the $M_a(x,y)$ case are as follows:

$$P_{M_a}(x,y) = \begin{cases} 1, & x \geq 1, y = 1, \\ 1, & x = 1, y \geq 1 \\ P_{M_a}(x-1,y) & \\ \quad + P_{M_a}(x,y-1) & \\ \quad\quad + P_{M_a}(x-1,y-1), & \text{otherwise} \end{cases} \tag{5.9}$$

$$L_{M_a}(x,y) = \begin{cases} x-1, & x \geq 1, y = 1, \\ y-1, & x = 1, y \geq 1 \\ L_{M_a}(x-1,y) & \\ \quad +L_{M_a}(x,y-1) & \\ \quad +L_{M_a}(x-1,y-1) & \\ \quad +P_{M_a}(x,y), & \text{otherwise} \end{cases} \tag{5.10}$$

As an example, we evaluate $P_{M_a}(x,y)$ and $L_{M_a}(x,y)$ with $x = 3, y = 2$. Using Eqns. (5.9) and (5.10), we have:

$$\begin{aligned} P_{M_a}(3,2) &= P_{M_a}(2,2) + P_{M_a}(3,1) + P_{M_a}(2,1) \\ &= P_{M_a}(2,2) + 2 \\ &= P_{M_a}(1,2) + P_{M_a}(2,1) + P_{M_a}(1,1) + 2 = 5 \end{aligned} \tag{5.11}$$

Thus, we have five effective paths with $\delta_{c_{1,1},c_{3,2}} \leq 3$, namely {(1,1),(2,2),(3,2)}, {(1,1),(2,1),(3,2)}, {(1,1),(1,2),(2,2),(3,2)}, {(1,1),(2,1),(3,1),(3,2)}, and {(1,1),(2,1),(2,2),(3,2)}.

$$\begin{aligned} L_{M_a}(3,2) &= L_{M_a}(2,2) + L_{M_a}(3,1) + L_{M_a}(2,1) \\ &\quad + P_{M_a}(3,2) \\ &= 5 + 2 + 1 + 5 = 13 \end{aligned} \tag{5.12}$$

Therefore, the ASD for $M_a(3,2)$ is $\mathcal{A}_{M_a}(x,y) = 13/5 = 2.6$.

Using generating functions described in [156] to solve the recurrence relations in Eqns. (5.9) and (5.10) for $x, y \geq 2$, [155] has shown that a generalized version is given by:

$$P_{M_a}(x,y) = \sum_{k=0}^{y-1} \binom{x-1}{k} \binom{x+y-k-2}{x-1} \tag{5.13}$$

For $x, y \geq 2$, we also have:

$$L_{M_a}(x,y) = (x-1)\binom{x-1}{y-1} + \sum_{k=0}^{y-2}\left[\binom{x-1}{k} \binom{x+y-k-2}{x} \frac{x(x+y-k-2)}{x-k-1}\right] \quad (5.14)$$

***Definition* 5.3:** *A function ${}_2\mathcal{F}_1(x,y;z;k)$ is a hypergeometric function if it can be expressed in the form:*

$${}_2\mathcal{F}_1(x,y;k;z) = \sum_{n=0}^{\infty} \frac{(x)_n (y)_n}{(k)_n} \frac{z^n}{n!} \quad (5.15)$$

Solving the Eqns. (5.13) and (5.14) analytically using Mathematica [157], we get:

$$P_{M_a}(x,y) = \binom{x+y-2}{x-1} F_1(x,y) \quad (5.16)$$

where $\mathcal{F}_1(x,y) = {}_2\mathcal{F}_1(1-x, 1-y; -x-y+2; -1)$. The first term in (5.14) can be written as:

$$(x-1)\binom{x-1}{y-1} = \frac{(x-1)\Gamma(x)}{\Gamma(y)\Gamma(x-y+1)} \quad (5.17)$$

where $\Gamma(\cdot)$ is the gamma function. With $x > y \geq 2$, the analysis in [155] shows that Eqn. (5.14) can be reduced to:

$$L_{M_a}(x,y) = \binom{x+y-2}{x-1} \left[(x+y-2)\mathcal{F}_1(x,y) - \frac{(x-1)(y-1)}{x+y-2}\mathcal{F}_2(x,y)\right] \quad (5.18)$$

where $\mathcal{F}_2(x,y) = {}_2\mathcal{F}_1(2-y; 2-x; -x-y+3; -1)$.

***Theorem* 5.2:** *For an augmented service mesh $M_a(x,y)$ of size $x \times y$ and $x, y \geq 1$, the*

average service distance $\mathcal{A}_{M_a}(1,1) = 0$. *Further,*

$$\mathcal{A}_{M_a}(x,y) = (x+y-2) - \left(\frac{(x-1)(y-1)}{x+y-2}\frac{\mathcal{F}_2(x,y)}{\mathcal{F}_1(x,y)}\right), \quad \forall x \geq y \geq 1, x, y \neq 1 \tag{5.19}$$

Lastly, when $y > x \geq 1, \mathcal{A}_{M_a}(x,y) = \mathcal{A}_{M_a}(y,x)$.

Proof. With $x, y = 1$, the result $\mathcal{A}_{M_a}(x,y) = 1$ follows from Eqns. (5.9) and (5.10). When $m \geq n \geq 1, m, n \neq 1$, we compute the average service distance defined in Eqn. (5.3) using Eqns. (5.13) and (5.14). For e.g., with $y = 3$, we see that:

$$\begin{aligned}\mathcal{F}_1(x,3) &= {}_2\mathcal{F}_1(1-x,-2;-x-1;-1) \\ &= \frac{2\left(2x^2-2x+1\right)}{x(x+1)}\end{aligned} \tag{5.20}$$

Also,

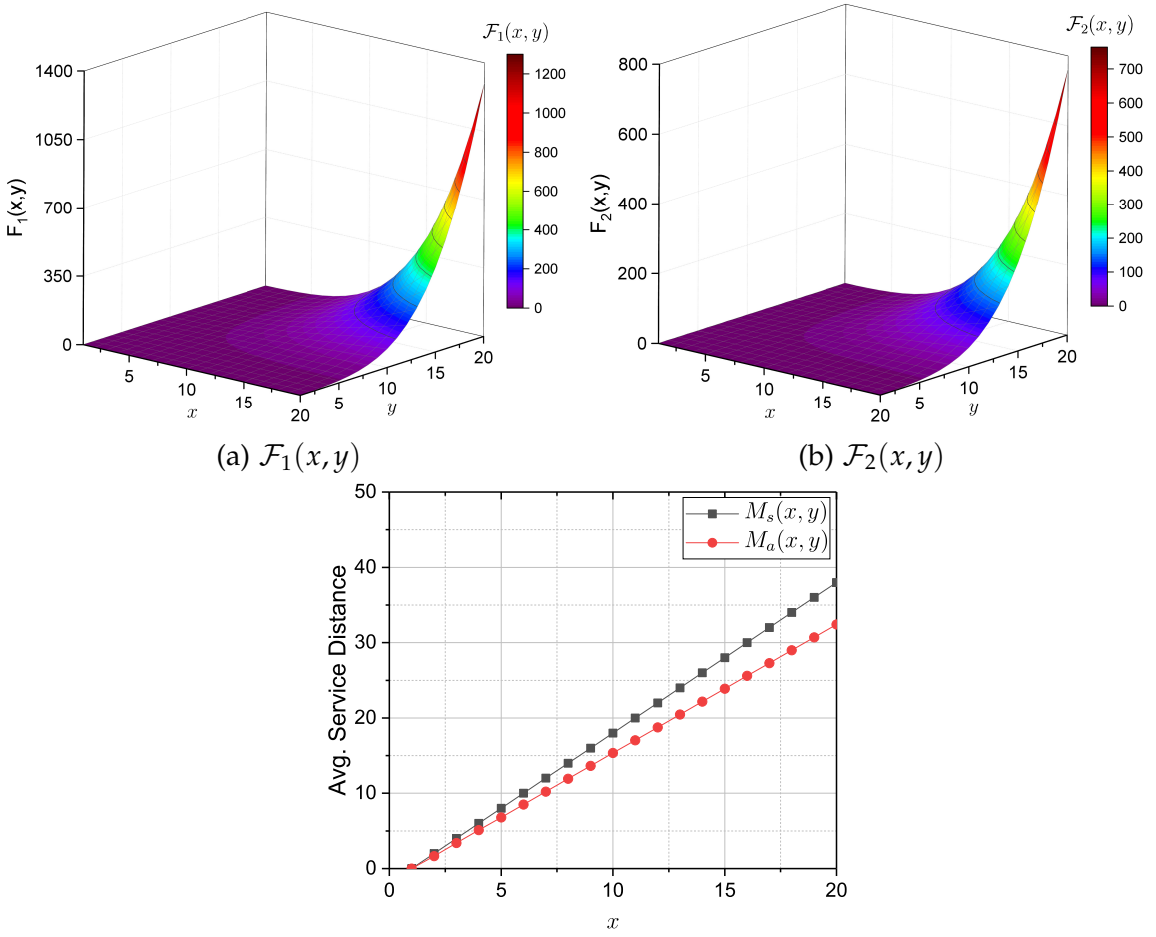
$$\mathcal{F}_2(x,3) = {}_2\mathcal{F}_1(-1;2-x;-x;-1) = \frac{2(x-1)}{x} \tag{5.21}$$

From Eqns. (5.13) and (5.14), it follows that

$$\begin{aligned}\mathcal{A}_{M_a}(x,3) &= (x+1) - \left(\frac{2(x-1)}{x+1}\frac{\mathcal{F}_2(x,y)}{\mathcal{F}_1(x,y)}\right) \\ &= \frac{2x^3-2x^2+3x-1}{2x^2-2x+1}\end{aligned} \tag{5.22}$$

From Eqns. (5.9) and (5.10), when $x \geq 1, y = 1$ we have $\mathcal{A}_{M_a}(x,y) = x-1$. To verify the last part of Theorem 5.2, we set $x = 1$ in (5.22). Thus, for $y > x \geq 1$, and $x = 1, y = 3$, we have $\mathcal{A}_{M_a}(1,3) = \mathcal{A}_{M_a}(3,1) = 2$. ■

Figures 5.9d, 5.9e and 5.9f show the variation of $P_{M_a}(x,y)$, $L_{M_a}(x,y)$ and $A_{M_a}(x,y)$ for $1 \leq x, y \leq 20$, respectively. From the figures, we observe that both



(a) $\mathcal{F}_1(x,y)$ (b) $\mathcal{F}_2(x,y)$

(c) Avg. Service Distance ($x = y$ Case)

Figure 5.10: Augmented service mesh performance and ASD comparison.

$P_{M_a}(x,y)$ and $L_{M_a}(x,y)$ grow exponentially with increasing service mesh size, albeit at a significantly faster rate than the simple two-dimensional service mesh. However, the presence of diagonal links in the service mesh leads to a modest decrease in the corresponding average service distance. The growth of $\mathcal{F}_1(x,y)$ and $\mathcal{F}_2(x,y)$ for $1 \leq x,y \leq 20$ are also shown in Figures 5.10a and 5.10b, respectively. Lastly, in Figure 5.10c, we present the comparison of ASD for both service mesh types, when $x = y$.

***Corollary* 5.2.1:** *When $x = y$, from Eqn. (5.19) it follows that*

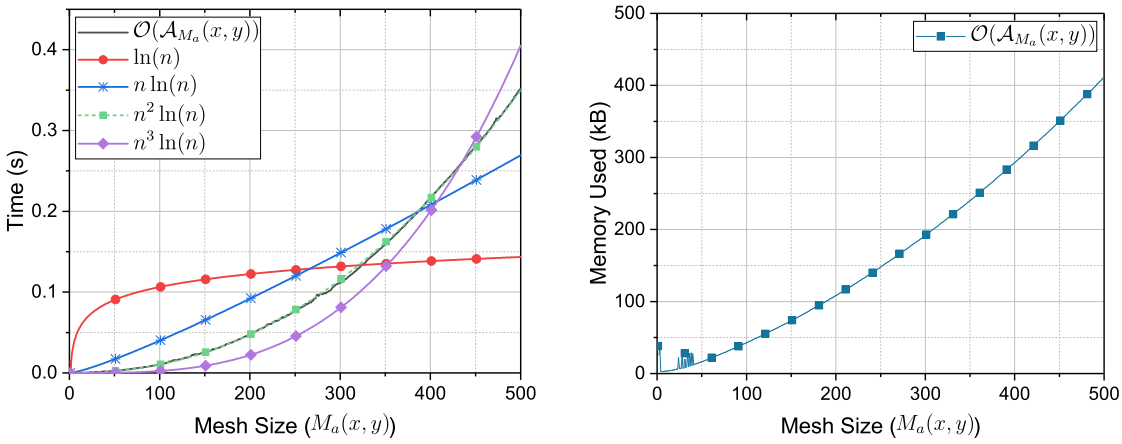
$$\mathcal{A}_{M_a}(x,x) = (2x-2) - \left(\frac{(x-1)}{2}\frac{\mathcal{F}_2(x,x)}{\mathcal{F}_1(x,x)}\right), \quad \forall x \geq 1, x \neq 1 \tag{5.23}$$

5.8.4 Average Service Distance Complexity

From Eqn. (5.8), it follows that the complexity of $\mathcal{A}_{M_s}(x,y)$ for service mesh $M_s(x,y)$ is $\mathcal{O}(n)$. The run-time complexity of the $M_a(x,y)$ service mesh to compute the average service distance ($\mathcal{A}_{M_a}(x,y)$) is shown in Figure 5.11. We show the $\mathcal{A}_{M_a}(x,y)$ computation time and space complexities in Figures 5.11a and 5.11b, respectively. Using non-linear regression to compute the best-fit approximations of the theoretical asymptotic run-time, we find that the time complexity is $\mathcal{O}(n^2\ln(n))$. The best-fit approximation for $\mathcal{A}_{M_a}(x,y), 1 \leq x,y \leq 500$ is $\approx 2.2515 \times 10^{-7} n^2 \ln(n)$. We also present the best-fit approximations for $\mathcal{O}(\ln(n))$, $\mathcal{O}(n\ln(n))$ and $\mathcal{O}(n^3\ln(n))$, which are $0.02309\ln(n)$, $8.6693 \times 10^{-5} n\ln(n)$, and $5.2143 \times 10^{-10} n^3 \ln(n)$, respectively, for comparison. In the following section, we develop a generalized service performance measure to reason over service mesh performance based on desired performance metrics.

5.9 Generalized Service Performance Measure

While the average service distance, $\mathcal{A}_M(x,y)$, provides a fundamental measure for understanding service performance across clusters, distributing services in real-world deployments often employ performance metrics that exhibit large variability across clusters. In Section 5.7, we assume a unit distance measure between services (i.e. $s^k_{i,j}$ and $s^k_{i\pm1,j\pm1}$) in adjacent clusters (i.e. $c_{i,j}$ and $c_{i\pm1,j\pm1}$) to



(a) Time Complexity (b) Space Complexity

Figure 5.11: $\mathcal{A}_{M_a}(x,y)$ Computational Complexity.

compute the average service distance for a given service mesh. However, service meshes in practice rely on numerous telemetry elements such as log information, performance metrics and distributed tracing to understand service behavior across clusters. Telemetry information can be context-aware (e.g., logs) or devoid of context due to personally identifiable information (PII) or unbounded cardinality. Uniform observability of services using various performance attributes is critical to ensure service resilience across diverse infrastructures. Performance studies have led to the development of numerous methods to identify critical metrics representative of system behavior. Examples include the USE method [158], four golden signals [159] and the RED method [160]. These methods rely on metrics such as utilization, error rates, saturation, latency, request rates and traffic to observe and monitor service performance.

We develop a generalized service performance measure (GSPM) to provide a metric(s)-based view of connected services in a network service mesh. We augment the average service distance using localized metrics representative of the service/workload performance within a given cluster. While the service/workload

performance metrics provide insight and ensure observability locally, the GSPM is vital to quantify and evaluate the acceptable end-to-end service performance upper bounds.

***Definition* 5.1:** *Let c_{i_1,j_1} and c_{i_2,j_2} denote a pair of arbitrary clusters in $M_a(x,y)$ located at (i_1,j_1) and (i_2,j_2), respectively. We define the transformed average service distance as:*

$$\begin{aligned}\mathcal{T}_{M_a}(i_1,j_1) =& \mathcal{A}_{M_a}(c_{i_1,j_1},c_{i_2,j_2}) \\ =& \mathcal{A}_{M_a}(c_{1,1},c_{t_1,t_2})\end{aligned} \tag{5.24}$$

where $t_1 = |i_2 - i_1| + 1$ and $t_2 = |j_2 - j_1| + 1$. Further, w.l.o.g, we note that $\mathcal{A}_{M_a}(c_{1,1},c_{t_1,t_2}) \equiv \mathcal{A}_{M_a}(t_1,t_2)$.

For each cluster, we can compute a transition matrix by using the transformation in Eqn. (5.24). Let $\mathbf{T}(i,j)$ denote the transformed average service distance transition matrix for the cluster $c_{i,j}$ at (i,j). Next, we integrate localized performance metrics with the above transition matrix to develop a generalized service performance measure for each cluster $c_{i,j} \in M_a(x,y)$.

***Definition* 5.2:** *Let $p_{i,j;t}$ denote a performance metric measured at time t on a service $s^k_{i,j}$ in a cluster $c_{i,j} \in M_a(x,y)$. We denote by $\mathbf{P}_M(x,y;t), x,y \in \mathbb{Z}, \forall t \in T$, a $x \times y$ matrix with $p_{i,j;t}$ for each cluster in $M_a(x,y)$. Then, we define the generalized service performance measure (GSPM) for the cluster $c_{i,j}$ as:*

$$\mathbf{G}(c_{i,j};t) = \mathbf{P}_M(i,j;t) \circ \mathbf{T}_{M_a}(i,j;t) \tag{5.25}$$

The product $\mathbf{P}_M(i,j;t) \circ \mathbf{T}_{M_a}(i,j;t)$ is the Hadamard product between the two

matrices at time t. Therefore, we have:

$$\begin{aligned}\mathbf{P}_M(i,j;t) \circ \mathbf{T}_{M_a}(i,j;t) &= (p_{i,j;t} \cdot \mathcal{T}_{i,j;t}) \\ &= \begin{pmatrix} p_{1,1;t} \cdot \mathcal{T}_{1,1;t} & \cdots & p_{1,j;t} \cdot \mathcal{T}_{1,j;t} \\ \vdots & \ddots & \vdots \\ p_{i,1;t} \cdot \mathcal{T}_{i,1;t} & \cdots & p_{i,j;t} \cdot \mathcal{T}_{i,j;t} \end{pmatrix}, \\ &\forall 1 \leq i \leq x, 1 \leq j \leq y, \forall t \in T\end{aligned} \tag{5.26}$$

Service meshes emphasize uniform observability, and therefore, we can use numerous performance metrics to compute the generalized service performance measure (GSPM). Typically, most cluster-wide metrics aggregators provide (hardware) resource usage information; however, they are limited to CPU, memory and storage resources. Apart from cluster-wide aggregated metrics, we can employ numerous other service performance indicators, including (i) service mesh metrics such as count and duration of requests, request/response size, TCP bytes sent/received, and TCP connections established/closed. (ii) application-specific metrics from the services' associated workloads, and (iii) a complex aggregated combination of the above. Unlike the average service distance, our proposed approach can easily integrate any of the above metrics to understand the distributed service performance across clusters better.

We present an algorithm to compute the aggregated generalized service performance measure (GSPM) in Algorithm 5.1. The algorithm considers performance metrics associated with a service ($s^k_{i,j} \in c_{i,j}$) or a set of workloads ($w^{k,l}_{i,j} \in s^k_{i,j}$) associated with a service for a given cluster $c_{i,j}$ at time t. For each service in a cluster, we first compute the transformed average service distance transition matrix (i.e., $\mathcal{T}_{M_a}(i,j;t)$). We then evaluate the GSPM matrix, $\mathbf{G}(c_{i,j};t)$, for the cluster $c_{i,j}$ at

time t using (5.25). $\mathbf{G}(c_{i,j}; t)$ provides a realistic measure of the expected service performance if we forward a request from the cluster $c_{i,j}$ to any other cluster $c_{i',j'}, i' \neq i, j' \neq j$ in the service mesh. In the above case, we assume a single unified metric associated with each service $s^k_{i,j}$.

Alternatively, we can also employ metrics associated with the services' workloads instead of a metric associated with it. However, to ensure scalability, workloads are typically load-balanced behind a given service. Thus, we have to aggregate the corresponding metrics from the services' workloads and combine them to a single representative value. In our proposed algorithm, we denote this aggregation function by $\mathcal{H}(\cdot)$. We note that an operator can develop numerous aggregation strategies based on application- and service-specific needs. Examples include strategies based on descriptive statistics or inference (including predictive) models.

***Lemma* 5.1:** *For a given metric $p_{i,j;t}$ measured at time t on a service ($s^k_{i,j}$) or aggregated from a set of workloads ($\{w^{k,l}_{i,j}\}$) in cluster $c_{i,j}$, the product $(p_{i,j;t} \cdot \mathcal{T}_{i,j;t})$ represents a bounded measure of the metric between two clusters.*

From Eqn. (5.24), we observe that the transformed average service distance, $\mathcal{T}_{i,j;t}$, measures the average distance between the source cluster at (i, j) and any other cluster in the service mesh. While the average service distance (ASD) indicates the normalized average distance between a pair of services (across clusters in the mesh), it does not account for localized performance characteristics. The product $p_{i,j;t} \cdot \mathcal{T}_{i,j;t}$, therefore, represents a bounded measure for a given service-pair weighted by the metric $p_{i,j;t}$ at time t.

***Theorem* 5.2:** *Let $\mathbf{P}_M(x, y; t), x, y \in \mathbb{Z}, \forall t \in T$ denote a $x \times y$ matrix with the metric $p_{i,j;t}$ measured at each cluster in $M_a(x, y)$. Then, the mesh-wide service performance*

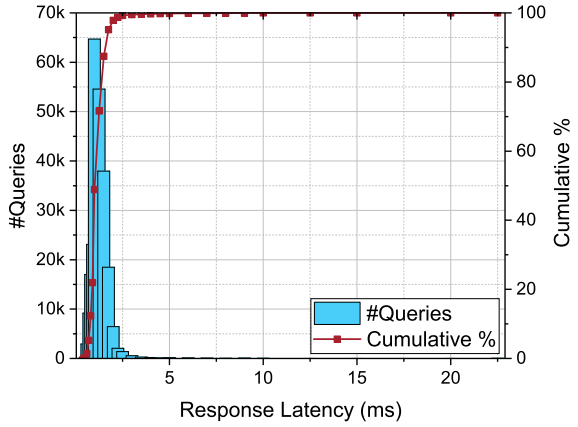
Algorithm 5.1 AggregatedGSPM($M(x,y), \mathcal{P}_M, \mathcal{A}_M(x,y)$

Require: Service Mesh $M(x,y)$, Performance metric $\mathbf{P}_M$, Transformed ASD $\mathcal{T}_M(x,y)$.
Output: Generalized Service Performance Matrix.

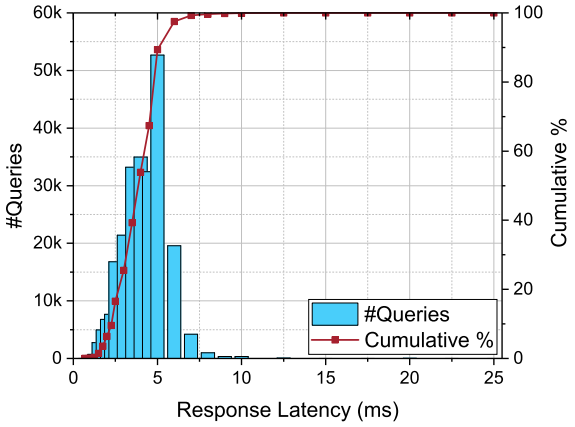
1: **for all** $c_{i,j} \in M_a(x,y)$ **do**
2: **for all** $s_{i,j}^k \in c_{i,j}$ **do**
3: **if** $p_{i,j;t} \mapsto s_{i,j}^k$ **then**
4: $\mathcal{T}_{M_a}(i,j;t) = \mathcal{A}_{M_a}(c_{i,j}, c_{i',j'}), i' \neq i, j' \neq j$
5: $\mathbf{T}_{M_a}(x,y;t) \leftarrow (\mathcal{T}_{M_a}(i,j;t))$
6: $\mathbf{G}(c_{i,j};t) = \mathbf{P}_M(x,y;t) \circ \mathbf{T}_{M_a}(x,y;t)$
7: **break**
8: **else if** $p_{i,j;t} \mapsto w_{i,j}^{k,l}$ **then**
9: **for all** $w_{i,j}^{k,l} \in s_{i,j}^k$ **do**
10: $\mathbf{P}_M(x,y;t) \leftarrow \mathcal{H}(\{p_{i,j;t}^{k,l}\}, w_{i,j}^{k,l})$
11: **go to** 4
12: **end for**
13: **end if**
14: **end for**
15: **end for**

as indicated by the metric $p_{i,j;t}$ is upper-bounded by the generalized service performance measure (GSPM), $\mathbf{G}(c_{i,j};t)$, for the cluster $c_{i,j}$.

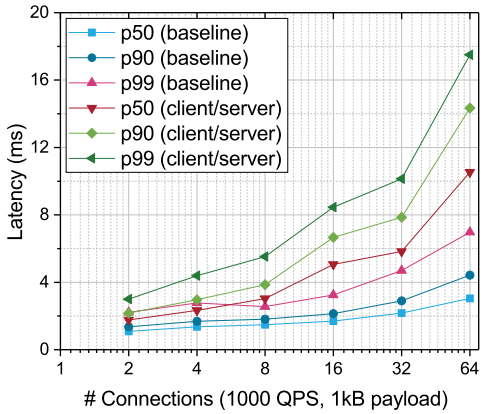
Proof. From Lemma 5.1, it follows that the Hadamard product $\mathbf{P}_M(x,y;t) \circ \mathbf{T}_{M_a}(x,y;t)$ represents a bounded measure for the service mesh weighted by the metric $p_{i,j;t}$ at time t. Thus, $\mathbf{G}(c_{i,j};t)$ is the mesh-wide upper bound for service performance as seen from the (source) cluster $c_{i,j}$. Further, as $\mathbf{T}_{M_a}(x,y;t)$ is weighted by $\mathbf{P}_M(x,y;t)$ with $c_{i,j}$ as the source, and therefore, the service performance cannot exceed $\mathbf{G}(c_{i,j};t)$ for predictable performance. ■



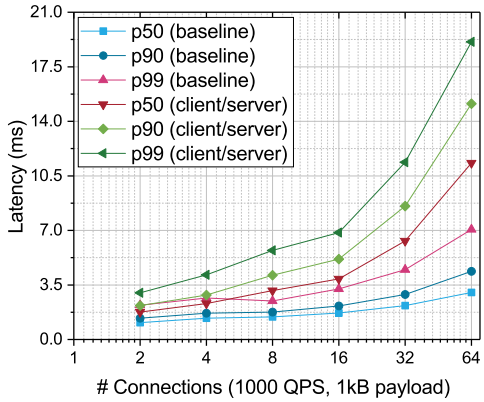
(a) Service Latency (Baseline)



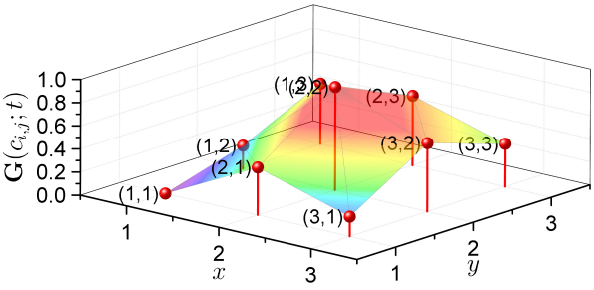
(b) Service Latency (Proxy/Sidecar)



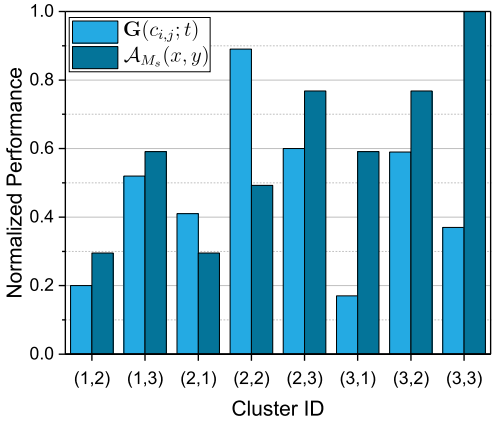
(c) Service Performance (No Encryption)



(d) Service Performance (mTLS)



(e) Normalized GSPM (Response Latency)



(f) $\mathbf{G}(c_{i,j};t)$ vs. $\mathcal{A}_{M_a}(i,j)$ (Normalized)

Figure 5.12: Prototype service mesh and GSPM-based dynamic routing performance.

5.10 Implementation

We build a prototype service mesh consisting of nine Kubernetes (K8s) [111, 161] clusters in a 3×3 configuration. Each of these nine K8s clusters are composed of three nodes with a single master node and two worker nodes. We deploy these clusters on an on-premise testbed that employs Dell PowerEdge R740 servers, with each server comprising of two Intel Xeon Gold 6126 processors (24 cores, 48 threads total), 192 GiB of RAM and 240 GiB SSDs. Each cluster in the service mesh is composed of 24 vCPUs and 24 GiB of RAM (8 vCPUs/8 GiB RAM per node). Further, we deploy an out-of-tree network file system (NFS) dynamic storage provisioner for managing persistent volumes on each cluster. Our service mesh infrastructure is built using the Istio [162] open source service mesh platform. Our service mesh employs an Istio multicluster deployment with replicated control planes. Services across clusters are connected using ingress/egress gateways that achieve intercluster service-to-service communications.

5.11 GSPM-based Dynamic Request Routing

Service meshes inherently provide the ability to gradually shift/transfer traffic from one service to another based on specified routing rules. Service-level properties of service mesh infrastructures enable service resiliency through the control/configuration of timeouts, circuit-breakers and retries. Further, weighed routing capabilities ensure ease of management for tasks such as blue/green deployments, staged/canary deployments, load balancing and A/B testing. While most service mesh framework implementations feature fundamental routing capabilities [163, 164] based on L4/L7 headers and service/application versioning, they rely on static configuration for routing traffic. In this section, we present a

GSPM-based routing technique to route incoming requests between intercluster services dynamically.

On each cluster of the service mesh described in Section 5.10, we deploy an identical directed acyclic service graph comprising four services: an API endpoint (S_A), a message broker (S_B), a task queue (S_C) and a worker (S_D). These four services form a service chain: $S_A \rightarrow S_B \rightarrow S_C \rightarrow S_D$. Further, using a routing policy, we configure the ingress/egress gateways to enable service request forwarding across clusters. For example, we can forward a request to the service S_B from S_A on $c_{1,1}$ to S_B on $c_{2,3}$. Natively, the service mesh supports service routing through a static, per-route weight assignment. For example, we can route 30%, 30%, and 40% of S_A traffic to $c_{1,2}, c_{2,1}$ and $c_{3,2}$. However, such an assignment fails to consider the current service performance at the corresponding cluster. Next, we demonstrate the effectiveness of GSPM in overcoming this problem to ensure optimum service routing based on current service performance metrics.

5.11.1 Results and Discussion

In Figure 5.12, we present the prototype service mesh performance. For the baseline, we evaluate service performance directly, i.e., without sidecars/proxies. Figures 5.12a and 5.12b show the service latency performance, with and without proxy/sidecars, respectively. We use a 1KB payload with 16 connections running 1000 queries/second (QPS) for a duration of 4 minutes. From the results, we observe that the service mesh proxies/sidecars contribute about $1 - 4$ms of latency per service-pair. We also show the service mesh latency for service-to-service communication with and without mTLS encryption in Figures 5.12c and 5.12d, respectively. We vary the number of connections from 2 to 64 using 1KB payloads at 1000QPS for 4 minutes. At the 90^{th} percentile, for 16 connections, we observe

that the service mesh infrastructure adds latencies of 4.52 ms and 3.01 ms over the baseline for unencrypted and mTLS (preferred) connections, respectively.

In Figure 5.12e, we present the normalized GSPM computed using response latency as the metric ($p_{i,j;t}$). A comparison with the average service distance for the 3×3 mesh with $c_{1,1}$ as the source is also presented in Figure 5.12f. Note that unlike GSPM or ASD, existing naive implementations reroutes traffic only during failures or circuit breaking. As shown in Figure 5.12f, ASD policy relies on intercluster latency only for choosing the cluster to forward service traffic. However, unlike ASD, our proposed GSPM approach takes localized service performance as indicated by a desired metric (response latency in our case) into account. Thus, our GSPM-based dynamic routing takes a holistic approach to service routing. By computing $\mathbf{G}(c_{i,j}; t)$ periodically, the routing policy ensures optimum service routing based on the desired performance metric. From the comparison, GSPM is $12.01 - 80.74\%$ better than ASD as an indicator of service performance measure. while ASD requires manual update. Lastly, periodic GSPM updates result in, on average, over 42.8% improvement of the accuracy of service routing performance over ASD.

5.12 Conclusions

In this work, we present ERGO, an edge computing architecture for Ag-IoT environments characterized by limited/intermittent internet connectivity. We develop edge-enabled Ag-IoT services that are modular, composable, and highly scalable in heterogeneous, resource-constrained environments. In particular, we show the challenges and opportunities arising from edge deployments in environments that cannot inherently offload data processing to cloud-based systems. By providing

localized, real-time decision-making tools, edge solutions can augment highly instrumented Ag-IoT environments. Our exemplary applications and extensive performance evaluations demonstrate the efficacy of our proposed architecture and outline numerous opportunities for developing optimized edge solutions for connectivity challenged Ag-IoT environments. In comparison to traditional architectures, on average, our proposed ERGO solution improves peak transaction throughput by over 77% and reduces response latencies by over 54%.

Further, we present a generalized service performance measure (GSPM) for multi-cluster distributed service mesh architectures. First, we presented the average service distance (ASD) measure to develop a fundamental framework for measuring service performance. We also presented a comprehensive analysis of two 2D service mesh structures: a simple service mesh and an augmented service mesh. Based on the shortcomings of ASD, we developed the generalized service performance measure (GSPM) to provide a metric-centric view of service performance by taking localized metrics that represent the service/workload performance, into account. We also demonstrate the effectiveness of our proposed GSPM-based dynamic routing approach by evaluating it on a real-world service mesh testbed and show that it performs, on average, 42.8% better than the alternative.

Chapter 6

Conclusions

In this dissertation, we investigate application-aware and software defined networking (SDN) solutions for network management problems, including service differentiation, quality of service (QoS), monitoring, service placement, policy enforcement, load balancing, network data analytics and service networking. First, we developed SNAG, an SDN-managed network architecture for data-intensive science workflows. SNAG is an exemplary solution for at-scale GridFTP traffic classification, monitoring, and management. Our solution demonstrates a cross-layer collaborative approach and integrates SDN with application-layer intelligence. Our application-aware approach effectively creates traffic classification and monitoring views that are not achievable using traditional layering approaches. Further, by exploiting application metadata at the network-layer, intelligent network management decisions are possible. At HCC, SNAG application-awareness played a crucial role in helping us understand the resource utilization patterns of opportunistic users (e.g. LIGO). Using application-awareness, network operators and resource owners can gain valuable insights and account for resources during opportunistic sharing. While SNAG focused on data-intensive science workflows and GridFTP integration, we can easily extend our application-aware architecture to securely expose metadata from other applications.

In Chapter 2, we also present an application-aware SDN approach to network services differentiation and active network management. Using this approach, we proposed a policy-based framework for network services differentiation. We demonstrated an application-driven mechanism for creating workflow-specific resource queues. Using application-aware network management principles, site operators can optimize resource allocation and fine-tune network performance to suit end-user application requirements. We also present two strategies, resource isolated queues (RIQ) and resource switched queues (RSQ), for applying QoS to CMS and LIGO flows.

In Chapter 3, we presented an integer linear programming formulation of the service function chain (SFC) mapping problem for multi-data center network topology. To reduce the SFC mapping costs across different data centers, we proposed a novel application-aware flow reduction (AAFR) algorithm. We present a thorough study of the SFC mapping problem by developing comprehensive models for both virtual network function (VNF) placemnt and SFC mapping across data centers. Through extensive performance evaluations of our proposed approach, we evaluate the performance of our AAFR algorithm and quantify the of impacts the application-aware flow processing on the number of SFCs and the SFC length on mapping costs. We also compare capacitated/uncapacitated cost gains, and finally investigate balancing flow-to-SFC mappings across data centers. Extensive performance evaluations show that our proposed AAFR algorithm achieves a maximum cost-gain of 70% for the capacitated SFC mapping case. Further, for the uncapacitated case, our AAFR algorithm provides an additional 4.1% cost-gain over its capacitated-SFC counterpart. Thus, the AAFR algorithm developed using application-aware principles is better at balancing flow-to-SFC mappings and avoids SFC loading problems.

In Chapter 4, we propose an application-aware intelligent load balancing system (APRIL) for high-throughput data-intensive science workflows such as CMS and LIGO. Using application-awareness, we obtained a dataset representing *800 million* GridFTP transfer connections from a major U.S. CMS Tier-2 site. We presented an extensive analysis of this dataset to identify long-term temporal dependencies between different user roles and workflow memberships. Using the insights from the data analysis, we leveraged deep learning techniques for time-series modeling to develop an application-aware predictive analytics system. We developed long short-term memory (LSTM) and gated recurrent units (GRU) based recurrent neural network (RNN) forecasting models. Our deep RNN predictive analytics system accurately forecasts GridFTP connection loads and performs better than other statistical/machine learning models such as ARIMA and multi-layer perceptrons (MLP).

We also developed a novel application-aware, predictive and intelligent load balancer, APRIL, that effectively integrates application metadata and load forecast information to maximize server utilization. Through extensive experiments, we demonstrated the effectiveness of APRIL by comparing it with an existing production Linux Virtual Server (LVS) cluster. Lastly, we presented a deployment strategy for integrating APRIL with the GridFTP ecosystem. The deployment includes scalable and modular components, including workflow task separation using the microservices architecture, model registry and an extensible services API. This approach allows for easy maintenance and monitoring of the APRIL system.

In Chapter 5, we present ERGO, a scalable edge computing architecture for Ag-IoT environments characterized by limited/intermittent internet connectivity. ERGO provides modular, composable, and highly scalable edge-enabled services for heterogeneous, resource-constrained Ag-IoT environments. Using ERGO, we

present the challenges and opportunities arising from edge deployments in environments that cannot inherently offload data processing to cloud-based systems. Thus, ERGO provides localized, real-time decision-making tools to augment highly instrumented Ag-IoT environments. We demonstrate the effectiveness of ERGO through extensive performance evaluations and exemplary applications. When compared to traditional architectures, on average, our proposed ERGO solution improves peak transaction throughput by over 77% and reduces response latencies by over 54%.

Lastly, we present a generalized service performance measure (GSPM) for multi-cluster distributed service mesh architectures. To develop a fundamental framework for measuring service performance, we first present the average service distance (ASD) measure. Next, we present a comprehensive performance analysis of two 2D service mesh structures: a simple service mesh and an augmented service mesh. We also developed the generalized service performance measure (GSPM) to provide a metric-centric view of service performance by taking localized metrics representing the service/workload performance. We also demonstrate the effectiveness of our proposed GSPM-based dynamic routing approach by evaluating it on a real-world service mesh testbed and show that it performs, on average, 42.8% better than the alternative.

While our application-awareness solutions have focused primarily on the GridFTP data transfers, data-intensive science and service mesh architectures, it can be easily extended to other application domains. Further, our approach can benefit other network management tasks including routing/forwarding, traffic analysis/redirection, resource provisioning, and QoS. Our future work will focus on creating adaptive and intelligent strategies for automated policy enforcement, threat-intelligence management, traffic engineering and heterogeneous

service-based architectures. We will develop and apply our application-awareness principles to other application domains, including cloud-native networking, network service mesh architectures and network security. We will also focus on developing novel resource allocation and resource management schemes that will consider a broader range of application metadata parameters.

Bibliography

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

[2] B. Han, V. Gopalakrishnan, L. Ji, and S. Lee, "Network function virtualization: Challenges and opportunities for innovations," *IEEE Communications Magazine*, vol. 53, no. 2, pp. 90–97, Feb 2015. 1

[3] I. Monga, E. Pouyoul, and C. Guok, "Software-Defined Networking for Big-Data Science - Architectural Models from Campus to the WAN," in *2012 SC Companion: High Performance Computing, Networking Storage and Analysis*, Nov 2012, pp. 1629–1635. 1

[4] G. Wang, T. E. Ng, and A. Shaikh, "Programming Your Network at Run-time for Big Data Applications," in *Hot Topics in Software Defined Networks*, ser. HotSDN '12. ACM, 2012, pp. 103–108. 1

[5] S. Jain, M. Khandelwal, A. Katkar, and J. Nygate, "Applying big data technologies to manage QoS in an SDN," in *2016 12th Conference on Network and Service Management (CNSM)*, Oct 2016, pp. 302–306. 1

[6] L. Cui, F. R. Yu, and Q. Yan, "When big data meets software-defined networking: SDN for big data and big data for SDN," *IEEE Network*, vol. 30, no. 1, pp. 58–65, January 2016. 1, 4.1

[7] D. Bonacorsi, "The CMS Computing Model," *Nuclear Physics B - Proceedings Supplements*, vol. 172, pp. 53 – 56, 2007. 1.1, 2.1, 2.2, 3.2.2, 4.1, 4.3.1

[8] G. Aad, J. Butterworth, J. Thion, U. Bratzler, P. Ratoff, R. Nickerson, J. Seixas, I. Grabowska-Bold, F. Meisel, S. Lokwitz *et al.*, "The ATLAS experiment at the CERN large hadron collider," *JINST*, vol. 3, p. S08003, 2008. 1.1

[9] W. Allcock, J. Bresnahan, R. Kettimuthu, and M. Link, "The Globus Striped GridFTP Framework and Server," in *Supercomputing, 2005. Proceedings of the ACM/IEEE SC 2005 Conference*, Nov 2005, pp. 54–54. 1.1, 3.5, 4.1

[10] A. Dorigo, P. Elmer, F. Furano, and A. Hanushevsky, "XROOTD-A Highly scalable architecture for data access," *WSEAS Transactions on Computers*, vol. 1, no. 4.3, 2005. 1.1, 2.5.2, 2.10, 4.1

[11] E. Dart, L. Rotman, B. Tierney, M. Hester, and J. Zurawski, "The Science DMZ: A network design pattern for data-intensive science," *Scientific Programming*, vol. 22, no. 2, pp. 173–185, 2014. 1.2, 1.2

[12] E. Balas and A. Ragusa, "SciPass: a 100Gbps capable secure Science DMZ using OpenFlow and Bro," in *Supercomputing 2014 Conference (SC14)*, 2014. 1.2, 1.2

[13] D. Nadig Anantha and B. Ramamurthy, "ScienceSDS: A Novel Software Defined Security Framework for Large-scale Data-intensive Science," in

ACM Security in Software Defined Networks & Network Function Virtualization, ser. SDN-NFVSec '17. ACM, 2017, pp. 13–18. 1.2, 1.3.3, 4.1

[14] S. Peisert, E. Dart, W. Barnett, E. Balas, J. Cuff, R. L. Grossman, A. Berman, A. Shankar, and B. Tierney, "The medical science DMZ: a network design pattern for data-intensive medical science," *Journal of the American Medical Informatics Association*, vol. 25, no. 3, pp. 267–274, 10 2017. 1.2, 1.2

[15] CDC, "HIPAA privacy rule and public health. Guidance from CDC and the US Department of Health and Human Services," *MMWR: Morbidity and mortality weekly report*, vol. 52, no. Suppl. 1, pp. 1–17, 2003. 1.2

[16] D. N. Anantha, Z. Zhang, B. Ramamurthy, B. Bockelman, G. Attebury, and D. Swanson, "SNAG: SDN-managed Network Architecture for GridFTP Transfers," in *3rd Workshop on Innovating the Network for Data-Intensive Science (INDIS) '16, SC16*, SLC, Utah, November 2016. 1.3.1, 1.4, 3.5, 3.7, 4.3.1, 4.4.1, 4.8.1

[17] D. N. Anantha, B. Ramamurthy, B. Bockelman, and D. Swanson, "Differentiated network services for data-intensive science using application-aware SDN," in *2017 IEEE ANTS*, Dec 2017, pp. 1–6. 1.3.2, 1.4, 4.3.1

[18] D. Nadig, B. Ramamurthy, B. Bockelman, and D. Swanson, "Identifying Anomalies in GridFTP Transfers for Data-Intensive Science Through Application-Awareness," in *Proceedings of the 2018 ACM International Workshop on Security in Software Defined Networks & Network Function Virtualization*, ser. SDN-NFV Sec'18. New York, NY, USA: ACM, 2018, pp. 7–12. 1.3.3

[19] D. Nadig, B. Ramamurthy, B. Bockelman, and D.Swanson, "Optimized Service Chain Mapping and reduced flow processing with Application-

Awareness," in *2018 4th IEEE Conference on Network Softwarization and Workshops (NetSoft)*, Jun. 2018, pp. 303–307. 1.4

[20] D. Nadig, B. Ramamurthy, B. Bockelman, and D. Swanson, "Large Data Transfer Predictability and Forecasting using Application-Aware SDN," in *2018 IEEE International Conference on Advanced Networks and Telecommunications Systems (ANTS)*, Dec. 2018, pp. 1–6. 1.4

[21] D. Nadig, B. Ramamurthy, B. Bockelman, and D.Swanson, "APRIL: An Application-Aware, Predictive and Intelligent Load Balancing Solution for Data-Intensive Science," in *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*, Apr. 2019, pp. 1909–1917. 1.4

[22] D. Nadig, S. El Alaoui, and B. Ramamurthy, "ERGO: A Scalable Edge Computing Architecture for Ag-IoT," in *3rd USENIX Workshop on Hot Topics in Edge Computing (HotEdge 20)*. USENIX Association, Jun. 2020. [Online]. Available: https://www.usenix.org/conference/hotedge20/presentation/nadig 1.4

[23] D. Nadig, S. El Alaoui, B. Ramamurthy, and S. Pitla, "ERGO: A Scalable Edge Computing Architecture for Infrastructureless Agricultural Internet of Things," in *2021 IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN) (LANMAN 2021), To appear*, Boston, USA, Jul. 2021. 1.4

[24] B. P. Abbott, R. Abbott, R. Adhikari *et al.*, "LIGO: the Laser Interferometer Gravitational-Wave Observatory," *Reports on Progress in Physics*, vol. 72, no. 7, p. 076901, 2009. 2.1, 3.2.2, 4.1

[25] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: Enabling Innovation in Campus

Networks," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, Mar. 2008. 2.1, 2.2.2

[26] C. Guok, E. N. Engineer, and D. Robertson, "Esnet On-demand Secure Circuits and Advance Reservation System (OSCARS)," *Internet2 Joint Techs*, 2006. 2.1, 2.2.1, 2.2.2

[27] W. Allcock, J. Bresnahan, R. Kettimuthu, and M. Link, "The Globus Striped GridFTP Framework and Server," in *Supercomputing, 2005. Proceedings of the ACM/IEEE SC 2005 Conference*, Nov 2005, pp. 54–54. 2.2

[28] W. Allcock, J. Bester, J. Bresnahan, A. Chervenak, L. Liming, and S. Tuecke, "GridFTP: Protocol Extensions to FTP for the Grid," in *Global Grid Forum*, 2003. 2.2

[29] R. Pordes, D. Petravick, B. Kramer, D. Olson, M. Livny, A. Roy, P. Avery, K. Blackburn, T. Wenaus, F. Wurthwein, I. Foster, R. Gardner, M. Wilde, A. Blatecky, J. McGee, and R. Quick, "The Open Science Grid," *Journal of Physics: Conference Series*, vol. 78, no. 1, p. 012057, 2007. 2.2

[30] E. Kissel, M. Swany, and A. Brown, "Improving GridFTP performance using the Phoebus session layer," in *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, Nov 2009, pp. 1–10. 2.2.1

[31] E. Kissel, M. Swany, and A. Brown, "Phoebus: A System for High Throughput Data Movement," *J. Parallel Distrib. Comput.*, vol. 71, no. 2, pp. 266–279, Feb. 2011. 2.2.1

[32] R. Tudoran, A. Costan, R. R. Rad, G. Brasche, and G. Antoniu, "Adaptive file management for scientific workflows on the Azure cloud," in *2013 IEEE International Conference on Big Data*, Oct 2013, pp. 273–281. 2.2.1

[33] R. Tudoran, A. Costan, and G. Antoniu, "Overflow: Multi-site aware big data management for scientific workflows on clouds," *IEEE Transactions on Cloud Computing*, vol. 4, no. 1, pp. 76–89, Jan 2016. 2.2.1

[34] D. Gunter, R. Kettimuthu, E. Kissel, M. Swany, J. Yi, and J. Zurawski, "Exploiting network parallelism for improving data transfer performance," in *2012 SC Companion: High Performance Computing, Networking Storage and Analysis*, Nov 2012, pp. 1600–1606. 2.2.1

[35] R. Kettimuthu, G. Agrawal, P. Sadayappan, and I. Foster, "Differentiated scheduling of response-critical and best-effort wide-area data transfers," in *2016 IEEE Parallel and Distributed Processing Symposium*, May 2016, pp. 1113–1122. 2.2.1

[36] E. Kissel and M. Swany, "The eXtensible Session Protocol: A Protocol for Future Internet Architectures," *Technical Report UDEL-2012/001*, 2012. 2.2.2

[37] B. Gibbard, D. Katramatos, D. Yu, and S. McKee, "TeraPaths: End-to-End Network Path QoS Configuration Using Cross-Domain Reservation Negotiation," in *3rd Intl. Conf. on Broadband Communications, Networks and Systems*, Oct 2006, pp. 1–9. 2.2.2

[38] D. Katramatos, D. Yu, B. Gibbard, and S. McKee, "The TeraPaths Testbed: Exploring End-to-End Network QoS," in *Conf. on Testbeds and Research Infrastructure for the Development of Networks and Communities*, May 2007, pp. 1–7. 2.2.2

[39] A. Tootoonchian, M. Ghobadi, and Y. Ganjali, "OpenTM: Traffic Matrix Estimator for OpenFlow Networks," in *Passive and Active Measurement*, A. Krishnamurthy and B. Plattner, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 201–210. 2.2.2

[40] N. 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)*, May 2014, pp. 1–8. 2.2.2

[41] W. Kim, J. Li, J. W. K. Hong, and Y. J. Suh, "OFMon: OpenFlow monitoring system in ONOS controllers," in *2016 IEEE NetSoft Conference and Workshops (NetSoft)*, June 2016, pp. 397–402. 2.2.2

[42] P. Berde, M. Gerola, J. Hart, Y. Higuchi, M. Kobayashi, T. Koide, B. Lantz, B. O'Connor, P. Radoslavov, W. Snow, and G. Parulkar, "ONOS: Towards an Open, Distributed SDN OS," in *3rd Workshop on Hot Topics in Software Defined Networking*, ser. HotSDN '14. New York, NY, USA: ACM, 2014, pp. 1–6. 2.4, 2.4.1

[43] Elastic stack. [Online]. Available: https://www.elastic.co/products 3, 2.5.3, 4.4.3

[44] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The Hadoop Distributed File System," in *Proceedings of the 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*, ser. MSST '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 1–10. 2.4.1, 2.5.1

[45] D. Nadig Anantha and B. Ramamurthy, "ScienceSDS: A Novel Software Defined Security Framework for Large-scale Data-intensive Science," in *ACM International Workshop on Security in Software Defined Networks & Network*

Function Virtualization, ser. SDN-NFVSec '17. New York, NY, USA: ACM, 2017, pp. 13–18. 2.4.1

[46] W. Allcock, J. Bresnahan, K. Kettimuthu, and J. Link, "The Globus eXtensible Input/Output System (XIO): A protocol independent IO system for the Grid," in *Parallel and Distributed Processing Symposium, 2005. Proceedings. 19th IEEE International*. IEEE, 2005, pp. 8–pp. 2.5.1

[47] I. Foster, *Globus Toolkit Version 4: Software for Service-Oriented Systems*. Springer Berlin Heidelberg, 2005, pp. 2–13. 2.5.1

[48] R. Egeland, T. Wildish, and C.-H. Huang, "PhEDEx Data Service," *Journal of Physics*, vol. 219, no. 6, 2010. 1

[49] W. Zhang, "Linux Virtual Server for Scalable Network Services," in *Ottawa Linux Symposium*, 2000. 2.6.2

[50] P. J. Brockwell and R. A. Davis, *Time Series: Theory and Methods*. Springer Science & Business Media, 2013. 2.7.2

[51] C. C. Holt, "Forecasting seasonals and trends by exponentially weighted moving averages," *International Journal of Forecasting*, vol. 20, no. 1, pp. 5 – 10, 2004. 2.7.2

[52] P. R. Winters, "Forecasting sales by exponentially weighted moving averages," *Management science*, vol. 6, no. 3, pp. 324–342, 1960. 2.7.2

[53] R. G. Brown, *Smoothing, Forecasting and Prediction of Discrete Time Series*. Courier Corporation, 2004. 2.7.2

[54] E. S. Gardner, "Exponential smoothing: The state of the art–Part II," *International Journal of Forecasting*, vol. 22, no. 4, pp. 637 – 666, 2006. 2.7.2

[55] D. N. Anantha, B. Ramamurthy, B. Bockelman, and D. Swanson, "Differentiated network services for data-intensive science using application-aware sdn," in *2017 IEEE International Conference on Advanced Networks and Telecommunications Systems (ANTS)*, Dec 2017, pp. 1–6. 2.8

[56] M. Devera. Hierarchical Token Bucket Theory. [Online]. Available: http://luxik.cdi.cz/~devik/qos/htb/manual/theory.htm 2.8.5

[57] 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," *SIGCOMM Comput. Commun. Rev.*, vol. 42, no. 4, pp. 13–24, Aug. 2012. 3.1

[58] J. G. Herrera and J. F. Botero, "Resource Allocation in NFV: A Comprehensive Survey," *IEEE Transactions on Network and Service Management*, vol. 13, no. 3, pp. 518–532, Sept 2016. 3.7

[59] R. Cohen, L. Lewin-Eytan, J. S. Naor, and D. Raz, "Near Optimal Placement of Virtual Network Functions," in *2015 IEEE Conference on Computer Communications (INFOCOM)*, April 2015, pp. 1346–1354. 3.7

[60] M. C. Luizelli, L. R. Bays, L. S. Buriol, M. P. Barcellos, and L. P. Gaspary, "Piecing together the NFV provisioning puzzle: Efficient placement and chaining of virtual network functions," in *2015 IFIP/IEEE Integrated Network Management (IM)*, May 2015, pp. 98–106. 3.7

[61] A. Hmaity, M. Savi, F. Musumeci, M. Tornatore, and A. Pattavina, "Virtual Network Function placement for resilient Service Chain provisioning," in *2016 8th Intl. Workshop on Resilient Networks Design and Modeling (RNDM)*, Sept 2016, pp. 245–252. 3.7

[62] M. Bouet, J. Leguay, and V. Conan, "Cost-based placement of vDPI functions in NFV infrastructures," in *2015 1st IEEE Conference on Network Softwarization (NetSoft)*, April 2015, pp. 1–9. 3.7

[63] T. V. Phan, N. K. Bao, Y. Kim, H.-J. Lee, and M. Park, "Optimizing resource allocation for elastic security VNFs in the SDNFV-enabled cloud computing," in *2017 Intl. Conference on Information Networking (ICOIN)*, Jan 2017, pp. 163–166. 3.7

[64] J. Liu, W. Lu, F. Zhou, P. Lu, and Z. Zhu, "On Dynamic Service Function Chain Deployment and Readjustment," *IEEE Transactions on Network and Service Management*, vol. PP, no. 99, pp. 1–1, 2017. 3.7

[65] H. Chen, S. Xu, X. Wang, Y. Zhao, K. Li, Y. Wang, W. Wang, and L. M. Li, "Towards optimal outsourcing of service function chain across multiple clouds," in *2016 IEEE International Conference on Communications (ICC)*, May 2016, pp. 1–7. 3.7

[66] A. M. Medhat, G. Carella, C. Luck, M.-I. Corici, and T. Magedanz, "Near optimal service function path instantiation in a multi-datacenter environment," in *2015 11th Intl. Conference on Network and Service Management (CNSM)*, Nov 2015, pp. 336–341. 3.7

[67] Z. Ye, X. Cao, J. Wang, H. Yu, and C. Qiao, "Joint topology design and mapping of service function chains for efficient, scalable, and reliable network functions virtualization," *IEEE Network*, vol. 30, no. 3, pp. 81–87, May 2016. 3.7

[68] L. Wang, Z. Lu, X. Wen, R. Knopp, and R. Gupta, "Joint Optimization of Service Function Chaining and Resource Allocation in Network Function Virtualization," *IEEE Access*, vol. 4, pp. 8084–8094, 2016. 3.7

[69] Z. A. Qazi, J. Lee, T. Jin, G. Bellala, M. Arndt, and G. Noubir, "Application-awareness in SDN," *SIGCOMM Comput. Commun. Rev.*, vol. 43, no. 4, pp. 487–488, Aug. 2013. 3.7

[70] S. Zhao, A. Sydney, and D. Medhi, "Building Application-Aware Network Environments Using SDN for Optimizing Hadoop Applications," in *2016 ACM SIGCOMM*. NY, USA: ACM, 2016, pp. 583–584. 3.7

[71] B. Wang, J. Su, L. Chen, J. Deng, and L. Zheng, "EffiEye: Application-aware Large Flow Detection in Data Center," in *17th IEEE/ACM Intl. Symposium on Cluster, Cloud and Grid Computing*, ser. CCGrid '17. Piscataway, NJ, USA: IEEE Press, 2017, pp. 794–796. 3.7

[72] D. N. Anantha and B. Ramamurthy, "ScienceSDS: A Novel Software Defined Security Framework for Large-scale Data-intensive Science," in *Proceedings of the ACM International Workshop on Security in Software Defined Networks & Network Function Virtualization*, ser. SDN-NFVSec '17. New York, NY, USA: ACM, 2017, pp. 13–18. 3.7

[73] D. Kreutz, F. M. V. Ramos, P. E. Veríssimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-Defined Networking: A Comprehensive Survey," *Proc. of the IEEE*, vol. 103, no. 1, pp. 14–76, Jan 2015. 4.1

[74] I. Monga, E. Pouyoul, and C. Guok, "Software-Defined Networking for Big-Data Science - Architectural Models from Campus to the WAN," in *2012*

SC Companion: High Performance Computing, Networking Storage and Analysis, Nov 2012, pp. 1629–1635. 4.1

[75] Y. Han, S.-s. Seo, J. Li, J. Hyun, J.-H. Yoo, and J. W.-K. Hong, "Software defined networking-based traffic engineering for data center networks," in *16th Asia-Pacific Network Operations and Management Symposium*, Sept 2014, pp. 1–6. 4.1

[76] S. Jain, M. Khandelwal, A. Katkar, and J. Nygate, "Applying big data technologies to manage QoS in an SDN," in *2016 12th Conference on Network and Service Management (CNSM)*, Oct 2016, pp. 302–306. 4.1

[77] G. Wang, T. E. Ng, and A. Shaikh, "Programming Your Network at Run-time for Big Data Applications," in *Hot Topics in Software Defined Networks*, ser. HotSDN '12. ACM, 2012, pp. 103–108. 4.1

[78] P. Qin, B. Dai, B. Huang, and G. Xu, "Bandwidth-Aware Scheduling With SDN in Hadoop: A New Trend for Big Data," *IEEE Systems Journal*, vol. 11, no. 4, pp. 2337–2344, Dec 2017. 4.1

[79] I. Monga, E. Pouyoul, and C. Guok, "Software-Defined Networking for Big-Data Science - Architectural Models from Campus to the WAN," in *2012 SC Companion: High Performance Computing, Networking Storage and Analysis*, Nov 2012, pp. 1629–1635. 4.1

[80] E. Dart, L. Rotman, B. Tierney, M. Hester, and J. Zurawski, "The Science DMZ: A network design pattern for data-intensive science," *Scientific Programming*, vol. 22, no. 2, pp. 173–185, 2014. 4.1

[81] D. Nadig and B. Ramamurthy, "Securing Large-Scale Data Transfers in Campus Networks: Experiences, Issues, and Challenges," in *Proceedings of the ACM International Workshop on Security in Software Defined Networks & Network Function Virtualization*, ser. SDN-NFVSec '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 29–32. 4.1

[82] D. Tuncer, M. Charalambides, S. Clayman, and G. Pavlou, "Adaptive Resource Management and Control in Software Defined Networks," *IEEE Trans. on Network and Service Management*, vol. 12, no. 1, pp. 18–33, March 2015. 4.1

[83] W. Jeong, G. Yang, S. M. Kim, and C. Yoo, "Efficient big link allocation scheme in virtualized software-defined networking," in *2017 13th Conf. on Network and Service Management (CNSM)*, Nov 2017, pp. 1–7. 4.1

[84] T. Zinner, M. Jarschel, A. Blenk, F. Wamser, and W. Kellerer, "Dynamic application-aware resource management using Software-Defined Networking: Implementation prospects and challenges," in *IEEE NOMS*, May 2014, pp. 1–6. 4.1

[85] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, p. 436, 2015. 4.1

[86] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997. 4.1, 4.5.1

[87] K. Cho, B. Van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using RNN encoder-decoder for statistical machine translation," in *Conference on Empirical Methods in Natural Language Processing*. Assn. for Computational Linguistics, 2014, pp. 1724–1734. 4.1, 4.5.1

[88] A. A. Neghabi, N. J. Navimipour, M. Hosseinzadeh, and A. Rezaee, "Load Balancing Mechanisms in the Software Defined Networks: A Systematic and Comprehensive Review of the Literature," *IEEE Access*, vol. 6, pp. 14 159–14 178, 2018. 4.2, 4.7

[89] M. Zhang and H. Yu, "A New Load Balancing Scheduling Algorithm Based on Linux Virtual Server," in *2013 Intl. Conf. on Computer Sciences and Applications*, Dec 2013, pp. 737–740. 4.2

[90] K. Wu, X. Wang, H. Chen, S. Zhao, and Y. Zhou, "Improvement on LVS based IP network connection status synchronization," in *IEEE Conference on Software Engineering and Service Science*, Sept 2015, pp. 746–749. 4.2

[91] J. Wang, J. Tang, Z. Xu, Y. Wang, G. Xue, X. Zhang, and D. Yang, "Spatiotemporal modeling and prediction in cellular networks: A big data enabled deep learning approach," in *IEEE INFOCOM 2017*, May 2017, pp. 1–9. 4.2

[92] P. Wang, S. Lin, and M. Luo, "A Framework for QoS-aware Traffic Classification Using Semi-supervised Machine Learning in SDNs," in *IEEE Conference on Services Computing*, June 2016, pp. 760–765. 4.2

[93] F. Tang, Z. M. Fadlullah, B. Mao, and N. Kato, "An Intelligent Traffic Load Prediction Based Adaptive Channel Assignment Algorithm in SDN-IoT: A Deep Learning Approach," *IEEE IoT Journal*, pp. 1–1, 2018. 4.2

[94] J. Xu, J. Wang, Q. Qi *et al.*, "IARA: An Intelligent Application-Aware VNF for Network Resource Allocation with Deep Learning," in *2018 15th IEEE SECON*, June 2018, pp. 1–3. 4.2

[95] D. Nadig, B. Ramamurthy, B. Bockelman, and D. Swanson, "Large Data Transfer Predictability and Forecasting using Application-Aware SDN," in *2018 IEEE International Conference on Advanced Networks and Telecommunications Systems (ANTS)*, 2018, pp. 1–6. 4.2

[96] W. Allcock, J. Bresnahan, R. Kettimuthu, and J. Link, "The Globus EXtensible Input/Output System (XIO): A Protocol Independent IO System for the Grid," in *Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05) - Workshop 4 - Volume 05*, ser. IPDPS '05. USA: IEEE Computer Society, 2005, p. 179.1. 4.3.1, 4.4.1

[97] P. J. Brockwell and R. A. Davis, *Introduction to Time Series and Forecasting*. springer, 2016. 4.3.2

[98] W. Zhang and W. Zhang, "Linux Virtual Server Clusters: Build highly-scalable and highly available network services at low cost," *Linux Magazine*, vol. 11, 2003. 4.4, 4.6.4, 4.7

[99] J. F. Kolen and S. C. Kremer, *Gradient Flow in Recurrent Nets: The Difficulty of Learning LongTerm Dependencies*, 2001, pp. 237–243. 4.5.1

[100] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 11 1997. 4.5.1

[101] F. Gers, J. Schmidhuber, and F. Cummins, "Learning to forget: continual prediction with LSTM," in *1999 Ninth International Conference on Artificial Neural Networks ICANN 99. (Conf. Publ. No. 470)*, vol. 2, 1999, pp. 850–855 vol.2. 4.5.1

[102] C. Laurent, G. Pereyra, P. Brakel, Y. Zhang, and Y. Bengio, "Batch normalized recurrent neural networks," in *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2016, pp. 2657–2661. 4.5.1

[103] J. L. Ba, J. R. Kiros, and G. E. Hinton, "Layer Normalization," 2016. 4.5.1

[104] K. Greff, R. K. Srivastava, J. Koutnik, B. R. Steunebrink, and J. Schmidhuber, "LSTM: A Search Space Odyssey," *IEEE Trans.Neural Networks and Learning Systems*, vol. 28, no. 10, pp. 2222–2232, Oct 2017. 4.5.1

[105] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014. 4.5.2, 4.6.2

[106] G. E. Box, G. M. Jenkins, G. C. Reinsel, and G. M. Ljung, *Time series analysis: forecasting and control*. John Wiley & Sons, 2015. 4.5.3

[107] N. Kratzke and P.-C. Quint, "Understanding cloud-native applications after 10 years of cloud computing - A systematic mapping study," *Journal of Systems and Software*, vol. 126, pp. 1–16, Apr. 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0164121217300018 5.1

[108] D. Gannon, R. Barga, and N. Sundaresan, "Cloud-Native Applications," *IEEE Cloud Computing*, vol. 4, no. 5, pp. 16–21, Sep. 2017, conference Name: IEEE Cloud Computing. 5.1

[109] 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*, M. Mazzara and B. Meyer, Eds. Cham: Springer International Publishing, 2017, pp. 195–216. [Online]. Available: https://doi.org/10.1007/978-3-319-67425-4_12 5.1

[110] J. Thönes, "Microservices," *IEEE Software*, vol. 32, no. 1, pp. 116–116, Jan. 2015, conference Name: IEEE Software. 5.1

[111] E. A. Brewer, "Kubernetes and the Path to Cloud Native," in *6th ACM Symposium on Cloud Computing*. ACM, 2015. 5.1, 5.3, 5.10

[112] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. D. Joseph, R. Katz, S. Shenker, and I. Stoica, "Mesos: a platform for fine-grained resource sharing in the data center," in *Proceedings of the 8th USENIX conference on Networked systems design and implementation*, ser. NSDI'11. USA: USENIX Association, Mar. 2011, pp. 295–308. 5.1

[113] E. Park, Y. Cho, J. Han, and S. J. Kwon, "Comprehensive Approaches to User Acceptance of Internet of Things in a Smart Home Environment," *IEEE IoT Journal*, vol. 4, no. 6, pp. 2342–2350, Dec 2017. 5.1

[114] L. Sanchez, L. Muñoz, J. A. Galache, P. Sotres, J. R. Santana, V. Gutierrez, R. Ramdhany, A. Gluhak, S. Krco, E. Theodoridis, and D. Pfisterer, "SmartSantander: IoT experimentation over a smart city testbed," *Computer Networks*, vol. 61, pp. 217–238, Mar. 2014. 5.1

[115] N. R. Sastra and D. M. Wiharta, "Environmental monitoring as an IoT application in building smart campus of Universitas Udayana," in *2016 International Conference on Smart Green Technology in Electrical and Information Systems (ICSGTEIS)*, Oct. 2016, pp. 85–88. 5.1

[116] S. M. R. Islam, D. Kwak, M. H. Kabir, M. Hossain, and K. Kwak, "The Internet of Things for Health Care: A Comprehensive Survey," *IEEE Access*, vol. 3, pp. 678–708, 2015. 5.1

[117] O. Elijah, T. A. Rahman, I. Orikumhi, C. Y. Leow, and M. N. Hindia, "An Overview of Internet of Things (IoT) and Data Analytics in Agriculture: Benefits and Challenges," *IEEE Internet of Things Journal*, vol. 5, no. 5, pp. 3758–3773, Oct. 2018. 5.1, 5.2

[118] C. Brewster, I. Roussaki, N. Kalatzis, K. Doolin, and K. Ellis, "IoT in Agriculture: Designing a Europe-Wide Large-Scale Pilot," *IEEE Communications Magazine*, vol. 55, no. 9, pp. 26–33, September 2017. 5.1

[119] S. Ivanov, K. Bhargava, and W. Donnelly, "Precision Farming: Sensor Analytics," *IEEE Intelligent Systems*, vol. 30, no. 4, pp. 76–80, Jul. 2015. 5.1

[120] J. A. Manrique, J. S. Rueda-Rueda, and J. M. Portocarrero, "Contrasting Internet of Things and Wireless Sensor Network from a Conceptual Overview," in *2016 IEEEiThings and IEEE GreenCom and IEEE CPSCom and IEEE SmartData*, Dec. 2016, pp. 252–257. 5.1

[121] P. Quinn, U. Elzur, and C. Pignataro, "Network service header (NSH)," in *RFC 8300*. RFC Editor, 2018. 5.1

[122] B. Pfaff, J. Pettit, T. Koponen, E. J. Jackson, A. Zhou, J. Rajahalme, J. Gross, A. Wang, J. Stringer, P. Shelar, K. Amidon, and M. Casado, "The Design and Implementation of Open VSwitch," in *Proceedings of the 12th USENIX Conference on Networked Systems Design and Implementation*, ser. NSDI'15. USA: USENIX Association, 2015, p. 117–130. 5.1

[123] "Tungsten Fabric," Available: https://tungsten.io, Last accessed on August 19, 2020. 5.1

[124] B. Dab, I. Fajjari, M. Rohon, C. Auboin, and A. Diquélou, "Cloud-native Service Function Chaining for 5G based on Network Service Mesh," in *ICC 2020 - 2020 IEEE International Conference on Communications (ICC)*, Jun. 2020, pp. 1–7, iSSN: 1938-1883. 5.2

[125] H.-L. Truong, L. Gao, and M. Hammerer, "Service architectures and dynamic solutions for interoperability of IoT, network functions and cloud resources," in *Proceedings of the 12th European Conference on Software Architecture: Companion Proceedings*, ser. ECSA '18. New York, NY, USA: Association for Computing Machinery, Sep. 2018, pp. 1–4. [Online]. Available: https://doi.org/10.1145/3241403.3241407 5.2

[126] R. Chandramouli and Z. Butcher, "Building Secure Microservices-based Applications Using Service-Mesh Architecture," 2020-05-27 2020. 5.2

[127] M. Kang, J.-S. Shin, and J. Kim, "Protected Coordination of Service Mesh for Container-Based 3-Tier Service Traffic," in *2019 International Conference on Information Networking (ICOIN)*, Jan. 2019, pp. 427–429, iSSN: 1976-7684. 5.2

[128] X. XIE and S. S. Govardhan, "A Service Mesh-Based Load Balancing and Task Scheduling System for Deep Learning Applications," in *2020 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGRID)*, May 2020, pp. 843–849. 5.2

[129] R. Loomba, T. Metsch, L. Feehan, and J. Butler, "A Hybrid Fitness-Utility Algorithm for Improved Service Chain Placement," in *2018 IEEE Global Communications Conference (GLOBECOM)*, Dec. 2018, pp. 1–7, iSSN: 2576-6813. 5.2

[130] G. Sun, Y. Li, D. Liao, and V. Chang, "Service Function Chain Orchestration Across Multiple Domains: A Full Mesh Aggregation Approach," *IEEE Transactions on Network and Service Management*, vol. 15, no. 3, pp. 1175–1191, Sep. 2018, conference Name: IEEE Transactions on Network and Service Management. 5.2

[131] H. Johng, A. K. Kalia, J. Xiao, M. Vuković, and L. Chung, "Harmonia: A Continuous Service Monitoring Framework Using DevOps and Service Mesh in a Complementary Manner," in *Service-Oriented Computing*, S. Yangui, I. Bouassida Rodriguez, K. Drira, and Z. Tari, Eds. Cham: Springer International Publishing, 2019, pp. 151–168. 5.2

[132] W. Li, Y. Lemieux, J. Gao, Z. Zhao, and Y. Han, "Service Mesh: Challenges, State of the Art, and Future Research Opportunities," in *2019 IEEE International Conference on Service-Oriented System Engineering (SOSE)*, Apr. 2019, pp. 122–1225, iSSN: 2642-6587. 5.2

[133] A. El Malki and U. Zdun, "Guiding Architectural Decision Making on Service Mesh Based Microservice Architectures," in *Software Architecture*, T. Bures, L. Duchien, and P. Inverardi, Eds. Cham: Springer International Publishing, 2019, pp. 3–19. 5.2

[134] Y. Fu, D. Li, P. Barlet-Ros, C. Huang, Z. Huang, S. Shen, and H. Su, "A Skewness-Aware Matrix Factorization Approach for Mesh-Structured Cloud Services," *IEEE/ACM Transactions on Networking*, vol. 27, no. 4, pp. 1598–1611, Aug. 2019, conference Name: IEEE/ACM Transactions on Networking. 5.2

[135] Y. Mao, L. K. Saul, and J. M. Smith, "IDES: An Internet Distance Estimation Service for Large Networks," *IEEE Journal on Selected Areas in Communications*,

vol. 24, no. 12, pp. 2273–2284, Dec. 2006, conference Name: IEEE Journal on Selected Areas in Communications. 5.2

[136] J. Zhu, P. He, Z. Zheng, and M. R. Lyu, "Online QoS Prediction for Runtime Service Adaptation via Adaptive Matrix Factorization," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 10, pp. 2911–2924, Oct. 2017, conference Name: IEEE Transactions on Parallel and Distributed Systems. 5.2

[137] R. Zhu, D. Niu, and Z. Li, "Robust web service recommendation via quantile matrix factorization," in *IEEE INFOCOM 2017 - IEEE Conference on Computer Communications*, May 2017, pp. 1–9. 5.2

[138] Y. Fu and X. Xiaoping, "Self-Stabilized Distributed Network Distance Prediction," *IEEE/ACM Transactions on Networking*, vol. 25, no. 1, pp. 451–464, Feb. 2017, conference Name: IEEE/ACM Transactions on Networking. 5.2

[139] B. Liu, D. Niu, Z. Li, and H. V. Zhao, "Network latency prediction for personal devices: Distance-feature decomposition from 3D sampling," in *2015 IEEE Conference on Computer Communications (INFOCOM)*, Apr. 2015, pp. 307–315, iSSN: 0743-166X. 5.2

[140] Y. Liao, W. Du, P. Geurts, and G. Leduc, "DMFSGD: a decentralized matrix factorization algorithm for network distance prediction," *IEEE/ACM Transactions on Networking*, vol. 21, no. 5, pp. 1511–1524, Oct. 2013. [Online]. Available: https://doi.org/10.1109/TNET.2012.2228881 5.2

[141] A. A. Bharate and M. S. Shirdhonkar, "A review on plant disease detection using image processing," in *2017 International Conference on Intelligent Sustainable Systems (ICISS)*, December 2017, pp. 103–109. 5.2

[142] M. Jhuria, A. Kumar, and R. Borse, "Image processing for smart farming: Detection of disease and fruit grading," in *2013 IEEE Second International Conference on Image Information Processing (ICIIP-2013)*, Dec. 2013, pp. 521–526. 5.2

[143] D. Shadrin, A. Menshchikov, D. Ermilov, and A. Somov, "Designing Future Precision Agriculture: Detection of Seeds Germination Using Artificial Intelligence on a Low-Power Embedded System," *IEEE Sensors Journal*, vol. 19, no. 23, pp. 11 573–11 582, Dec. 2019. 5.2

[144] A. Khattab, A. Abdelgawad, and K. Yelmarthi, "Design and implementation of a cloud-based IoT scheme for precision agriculture," in *2016 28th Intl. Conf. on Microelectronics (ICM)*, Dec. 2016, pp. 201–204. 5.2

[145] M. S. Mekala and P. Viswanathan, "A Survey: Smart agriculture IoT with cloud computing," in *2017 International conference on Microelectronic Devices, Circuits and Systems (ICMDCS)*, Aug. 2017, pp. 1–7. 5.2

[146] Federal Communications Commission (FCC), "2018 broadband deployment report," February 2018, [Online; posted 02-February-2018]. 5.2

[147] T. A. A. Ali, V. Choksi, and M. B. Potdar, "Precision Agriculture Monitoring System Using Green Internet of Things (G-IoT)," in *2018 2nd International Conference on Trends in Electronics and Informatics (ICOEI)*, May 2018, pp. 481–487. 5.2

[148] A. Javed, K. Heljanko, A. Buda, and K. Främling, "CEFIoT: A fault-tolerant IoT architecture for edge and cloud," in *2018 IEEE 4th World Forum on Internet of Things (WF-IoT)*, Feb. 2018, pp. 813–818. 5.2

[149] X. Chen, Q. Shi, L. Yang, and J. Xu, "ThriftyEdge: Resource-Efficient Edge Computing for Intelligent IoT Applications," *IEEE Network*, vol. 32, no. 1, pp. 61–65, Jan. 2018. 5.2

[150] "Flask (A Python Microframework)," https://flask.palletsprojects.com/, accessed: 2021-04-12. 5.3.2

[151] "MongoDB," https://www.mongodb.com/, accessed: 2021-04-12. 5.3.2

[152] J. P. Werner, "Flex-Ro: Design, Implementation, and Control of Subassemblies for an Agricultural Robotic Platform," Master's thesis, University of Nebraska-Lincoln, Lincoln, NE, USA, 2016. 5.5.2

[153] J. N. Murman, "Flex-Ro: A Robotic High Throughput Field Phenotyping System," Master's thesis, University of Nebraska-Lincoln, Lincoln, NE, USA, 2019. 5.5.2

[154] "Apache JMeter," https://jmeter.apache.org/, accessed: 2021-04-12. 5.6

[155] Z. Shen, "The calculation of average distance in mesh structures," *Computers & Mathematics with Applications*, vol. 44, no. 10, pp. 1379–1402, Nov. 2002. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S089812210200264X 5.8.2, 5.8.2, 5.8.3, 5.8.3

[156] R. L. Graham, D. E. Knuth, and O. Patashnik, *Concrete Mathematics: A Foundation for Computer Science*, 2nd ed. USA: Addison-Wesley Longman Publishing Co., Inc., 1994. 5.8.2, 5.8.3

[157] W. R. Inc., "Mathematica, Version 12.1," champaign, IL, 2020. [Online]. Available: https://www.wolfram.com/mathematica 5.8.3

[158] B. Gregg, "Thinking Methodically about Performance," *Queue*, vol. 10, no. 12, pp. 40–51, Dec. 2012. [Online]. Available: https://doi.org/10.1145/2405116.2413037 5.9

[159] B. Beyer, C. Jones, J. Petoff, and N. R. Murphy, *Site Reliability Engineering: How Google Runs Production Systems*, 1st ed. O'Reilly Media, Inc., 2016. 5.9

[160] T. Wilkie, "The RED Method: key metrics for microservices architecture," library Catalog: www.weave.works. [Online]. Available: https://www.weave.works 5.9

[161] B. Burns, B. Grant, D. Oppenheimer, E. Brewer, and J. Wilkes, "Borg, Omega, and Kubernetes," *Queue*, vol. 14, no. 1, p. 70–93, Jan. 2016. 5.10

[162] "Istio," Available: https://istio.io/, Last accessed on August 09, 2020. 5.10

[163] K. Indrasiri and P. Siriwardena, "Service Mesh," in *Microservices for the Enterprise: Designing, Developing, and Deploying*. Berkeley, CA: Apress, 2018, pp. 263–292. 5.11

[164] M. Klein, "Lyft's Envoy: Experiences Operating a Large Service Mesh," in *SRECon17 Americas*. San Francisco, CA: USENIX Association, Mar. 2017. 5.11