Graduate School Form
30 Updated 12/26/2015

PURDUE UNIVERSITY
GRADUATE SCHOOL
Thesis/Dissertation Acceptance

This is to certify that the thesis/dissertation prepared

By Harold Owens II

Entitled

PROVISIONING END-TO-END QUALITY OF SERVICE FOR REAL-TIME INTERACTIVE VIDEO OVER SOFTWARE-DEFINED NETWORKING

For the degree of Doctor of Philosophy

Is approved by the final examining committee:

Arjan Durresi
Chair

Mohammad Al Hasan

Xia N. Ning

Raje R. Rajeev

To the best of my knowledge and as understood by the student in the Thesis/Dissertation Agreement, Publication Delay, and Certification Disclaimer (Graduate School Form 32), this thesis/dissertation adheres to the provisions of Purdue University's "Policy of Integrity in Research" and the use of copyright material.

Approved by Major Professor(s): Arjan Durresi

Approved by: Shiaofen Fang 11/22/2016
Head of the Departmental Graduate Program Date

PROVISIONING END-TO-END QUALITY OF SERVICE FOR REAL-TIME INTERACTIVE VIDEO OVER SOFTWARE-DEFINED NETWORKING

A Dissertation

Submitted to the Faculty

of

Purdue University

by

Harold Owens II

In Partial Fulfillment of the

Requirements for the Degree

of

Doctor of Philosophy

December 2016

Purdue University

Indianapolis, Indiana

To my mother (1947-2006). To my wife, daughter, and son. Thank you.

ACKNOWLEDGMENTS

I want to acknowledge my mentor and advisor, Dr. Arjan Durresi for his encouragement and guidance during my studies. I want to acknowledge thesis committee members Dr. Rajeev Raje, Dr. Xia Ning, and Dr. Mohammad Al Hasan for reviewing and providing feedback on my thesis.

TABLE OF CONTENTS

Page

LIST OF TABLES

LIST OF FIGURES

Figure Page

ABSTRACT

Owens II, Harold. Ph.D., Purdue University, December 2016. Provisioning End-To-End Quality of Service for Real-Time Interactive Video over Software-Defined Networking. Major Professor: Arjan Durresi.

This thesis contains four interrelated research areas. Before presenting the four research areas, this thesis presents literature review on Software-Defined Networking (SDN), a network architecture that allows network operator to manage the network using high level abstractions. This thesis presents taxonomy for classifying SDN research.

In research first area, this thesis presents Video over Software-Defined Networking (VSDN), a network architecture that selects feasible paths using the network-wide view. This thesis describes the VSDN protocol which is used for requesting service from the network. This thesis presents the results of implementing VSDN prototype and evaluates behavior of VSDN. Requesting service from the network requires developer to provide three input parameters to application programmable interface. The message complexity of VSDN is linear.

In research second area, this thesis presents Explicit Routing in Software-Defined Networking (ERSDN), a routing scheme that selects transit routers at the edge of network. This thesis presents the design and implementation of ERSDN. This thesis evaluates the effect of ERSDN on the scalability of controller by measuring the control plane network events—packets. ERSDN reduces the network events in the control plane by 430%.

In research third area, this thesis presents Reliable Video over Software-Defined Networking (RVSDN) which builds upon previous work of Video over Software-Defined Networking (VSDN) to address the issue of finding most reliable path. This

thesis presents the design and implementation of RVSDN. This thesis presents the experience of integrating RVSDN into ns-3, a network simulator which research community uses to simulate and model computer networks. This thesis presents RVSDN results and analyzes the results. RVSDN services 31 times more requests than VSDN and Multiprotocol Label Switching (MPLS) explicit routing when the reliability constraint is 0.995 or greater.

In research fourth area, this thesis presents Multi-Domain Video over Software-Defined Networking (MDVSDN), a network architecture that selects end-to-end network path or path for real-time interactive video applications across independent network domains. This thesis describes the architectural elements of MDVSDN. This thesis presents the results of implementing a prototype of MDVSDN and evaluates the behavior of MDVSDN. The message complexity of MDVSDN is linear.

The contribution of this thesis lays the foundation for developing a network architecture that improves the performance of real-time interactive video applications by selecting feasible end-to-end multi-domain path among multiple paths using bandwidth, delay, jitter, and reliability.

1 INTRODUCTION

"The Holy Grail of computer networking is to design a network that has the flexibility and low cost of the Internet, yet offers the end-to-end quality-of-service guarantees of the telephone network." Srinivasan Keshav

1.1 Overview of Problem

Globally, the Internet Protocol (IP) video traffic such as video on demand (VOD) and interactive video makes up 67% of IP traffic [1]. The real-time interactive video applications require end-to-end quality of service (QoS) from the network. The real-time interactive video applications such as videoconferencing—Google Hangouts and Microsoft Skype and distance learning require guaranteed bandwidth, bounded delay, and bounded jitter from the network. The real-time interactive video applications such as telesurgery—remote surgery requires network reliability as well as guaranteed bandwidth, bounded delay, and bounded jitter. The computer network QoS frameworks are unable to meet the needs of real-time interactive video applications which require flow-based end-to-end QoS from the network.

There are four network QoS frameworks that provide QoS for video applications. Asynchronous Transfer Mode (ATM) provides end-to-end QoS for real-time application such as video, but ATM for various reasons such as cost, advent of Ethernet, complex software flow state setup process, and lack of integration standards was not widely adopted by network community [2]. Integrated Services (IntServ), a flow-based network architecture, provides end-to-end QoS for real-time and mission critical applications such as video and voice, but IntServ lacks scalability because the soft states in network devices need refreshing. IntServ uses message flooding to refresh the

state of network device. IntServ is unable to explore network paths that differs from the routing protocol shortest constraint path. Differentiated Services (DiffServ) addresses IntServ scalability issue using the class based approach where IP prefixes are aggregated into different classes, increasing network scalability while losing control over individual network flows. Multiprotocol Label Switching (MPLS), a label switch technique, increases packet switching speed in the core network using hashing and provides link failover in case of network failures. MPLS requires network operator to preconfigure links and MPLS lacks real-time path configuration. MPLS is unable to reject flows and MPLS is unable to guarantee bandwidth and bounded delay.

The QoS frameworks are unable to meet real-time interactive video application needs. This thesis seeks to develop network architecture that meets real-time interactive video application needs. This work builds on Software-Defined Networking (SDN), a network architecture that increases network programmability. SDN provides network applications such as traffic engineering and load balancing and services such as service chaining and wide-area network optimization with programmable network abstractions, network-wide state, and network feedback that allows the network applications and services to make routing decisions.

The network applications and services provide end-to-end QoS for real-time interactive video applications. The network applications and services use feedback from the network to make decisions including selecting an end-to-end network path or path. Selecting a path is difficult using the network QoS frameworks. The application-aware network applications—application that keeps track of application-level characteristics and network state and use information to provision end-to-end QoS paths are difficult to design and develop using the network QoS frameworks.

1.2 Motivation

"Constraint based routing evolves from QoS Routing. Given QoS request of a flow or an aggregation of flows, QoS Routing returns route that is most likely to be able to meet QoS requirements. Constraint based routing extends QoS Routing by considering other constraints of network such as policy." Xipeng Xiao and Lionel M. Ni

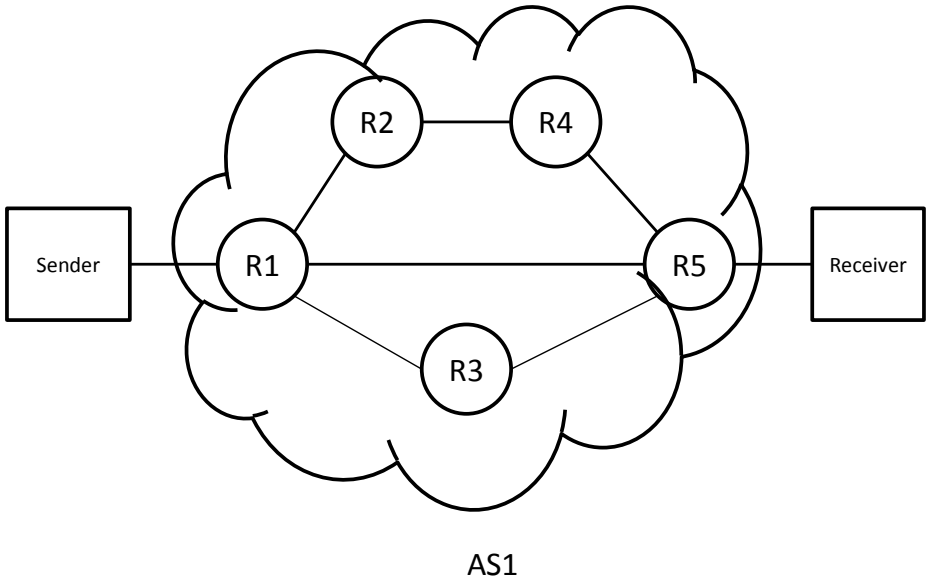

Figure 1.1.: Seven-node IP network with sender and receiver. The routers in autonomous systems one (AS1) are running Open Shortest Path First routing protocol which determines shortest path between the sender and receiver. The shortest path is R1-R5, but best path is R1-R2-R4-R5. The routers are unable to select feasible—best path.

Example. The network operator uses a real-time interactive video application with QoS constraints—bandwidth, delay, and jitter. The QoS requirements are not conceived beforehand. The network receives QoS request and provisions network resources—path.

In Figure 1.1, the best path for real-time interactive video application is R1-R2-R4-R5. The network uses IntServ reservation protocols with Open Shortest Path First (OSPF) routing protocol. If the sender and receiver start a video session, the network uses shortest path selected by OSPF—R1-R5; therefore, the video packets traverse path R1-R5 which is two hops. IntServ reservation protocol installs QoS—PATH and RESV states in R1 and R5; however, the best path for the video application is path R1-R2-R4-R5.

IntServ reservation protocols make it difficult to select a path that differs from the shortest path selected by routing protocols—Routing Information Protocol (RIP), Intermediate System to Intermediate System (IS-IS) , or OSPF because the routing protocols select the shortest path for forwarding packets [3]. In Figure 1.1, if link failure—R1-R5 occurs, the network finds another path. In case of a failure, OSPF discovers next shortest path which is path R1-R3-R5, in Figure 1.1. After 30 seconds, IntServ detects failure and provisions another path—R1-R3-R5 which PATH and RESV states are stored at each router. The best path for video application is path R1-R2-R4-R5. The network has failed to find a feasible path that satisfies the real-time interactive video application constraints.

Selecting a feasible path for real-time interactive video applications require the network architecture to select a path among multiple paths to improve video application performance [4]. The network architecture requires knowledge about the state of network such as bandwidth, link congestion, and link and node failures. If the network architecture has the network-wide view—complete view, the network architecture can select a feasible end-to-end path for the real-time interactive video application, in Figure 1.1.

There are design requirements needed for the network architecture to select a feasible end-to-end path for real-time interactive video applications. The requirements to select a feasible end-to-end path are.

- The network applications and services need the network-wide view—need centralized control to keep track of resources such as bandwidth.

- The network applications and services need the ability to program network behavior.
- The network architecture should reject request if the network is unable to service the request [2].
- The network architecture needs a traffic engineering (TE) service to allow network applications and services to program how traffic flows through the network [3].
- The network architecture needs to perform constraint based routing using bandwidth, delay, jitter, and reliability.
- The rate of network traffic should be known in advance [2].
- The network architecture should consistently enforce network policies and account for other network policies and QoS requirements of video application [5–8].
- The network architecture needs to support multi-domain end-to-end path selection since the Internet requires a collection of independent network domains to work together to provide QoS for video applications.

The network QoS architecture should incorporate traffic engineering (TE) with constraint based routing [3] and should know the traffic rate in advance [2]. Constraint based routing is an important tool for making TE process automatic [3]. Since traffic rate is known in advance, the output port and intermediate network devices can record the level of guaranteed traffic and reject flows if agreed capacity is in use [2]. The network QoS architecture should keep track of network resources including capacity levels and reject request if the network architecture is unable to service the request [9, 10].

1.3 Proposed Solution

"A motivation for Internet traffic engineering is the realization that architectural paradigms and simple capacity expansion are necessary, but not sufficient, to deliver high quality Internet service under all circumstances." Daniel O. Awduche

Internet service providers develop scalable network architectures, expand network capacity and network infrastructure, and perform traffic engineering in response to large traffic growth [11]. Traffic engineering optimizes network performance [11] and improves network operations efficiency and reliability [12].

In this thesis, traffic engineering is an integral part of developing a network architecture that provides end-to-end QoS for real-time interactive video applications.

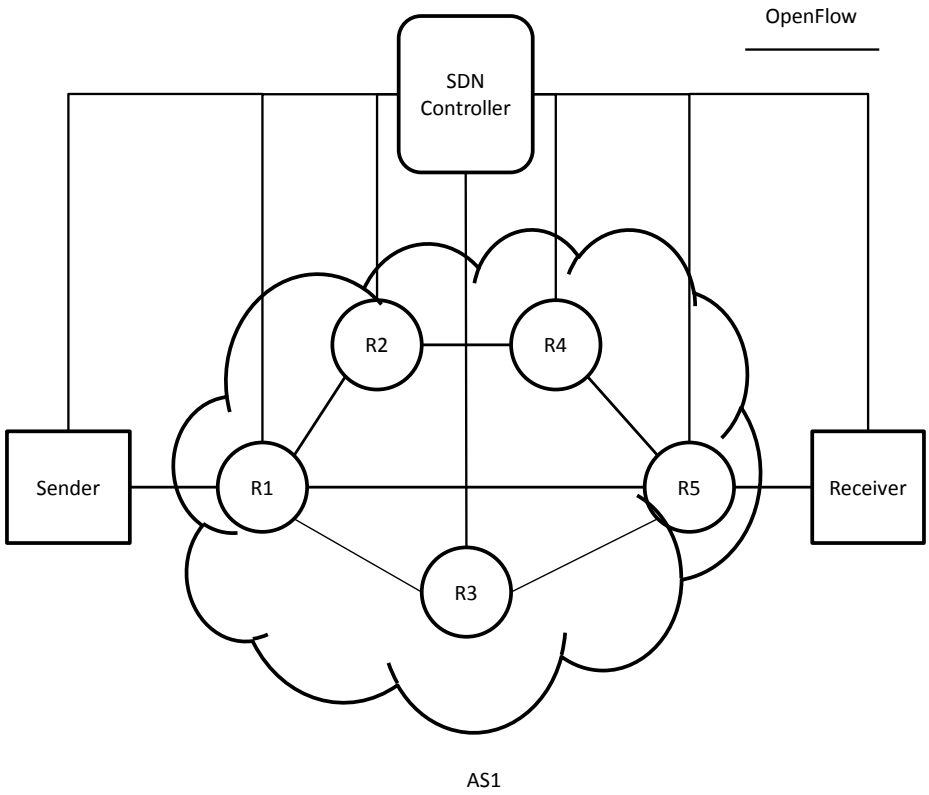

Figure 1.2.: SDN network with sender and receiver. The routers in AS1 are commodity network devices that the SDN controller programs for forwarding network traffic. The SDN controller has the network-wide view of network resources such as link state, congestion, bandwidth, delay, and jitter. The SDN controller selects a feasible path—R1-R2-R4-R5 using the network-wide view and programs network devices including the sender and receiver.

Software-defined networking (SDN) supports the proposed network architecture design requirements by.

- Giving network applications and services control of network behavior using high level abstractions to hide network details and to support network programmability
- Providing a centralized view of network resources
- Supporting centralized management of network resources
- Allowing network services and applications to direct and automate network traffic using TE and constraint based routing
- Allowing network policies to be applied consistently using the network-wide view

Although SDN allows network application and services to have the network-wide view, SDN lacks a provisioning protocol that requests QoS from the network and network service which provides QoS for real-time interactive video applications; therefore, this thesis develops a provisioning protocol and network service that meets the needs of real-time interactive video applications.

The provisioning protocol allows the applications to request QoS from the network. The network service provisions network resources—end-to-end path for the video application. The network architecture should reject requests the network is unable to service [2]. The network architecture supports multi-domain path selection.

The SDN architecture supports one network domain; therefore, the proposed network architecture supports multiple domains. At the higher-level architecture level, new concepts and constructs are required for dealing with end-to-end flows that involve multiple scarce resources [13]. This thesis presents the proposed network architecture key concepts and constructs and the results of implementing the proposed network architecture prototype.

1.4 Research Goals

"In order to adopt TE solutions, it is necessary to create an intelligent control plane which is able to adequately handle network resources." Roberto Sabella and Paola Iovanna

The research goal of this thesis is to develop a network architecture that meets the needs of real-time interactive video applications —automating TE process. This thesis proposes a network architecture that uses SDN as its core. The proposed network architecture takes advantage of the logically centralized control plane and network resource monitoring ability of SDN.

The primary research question of this thesis can be summarized as:

> What is the network architecture needed to support real-time interactive video applications?

As this thesis explored the primary research question other fundamental questions that relates to the primary research question were produced.

- What protocol is required to allow video applications to request QoS from the network?
  - Can the protocol accept minimal information from developer and meet video application requirements?
  - How difficult is it for video application developer to request QoS from the network using the application program interface (API)–API usability?
- How does the network architecture control state distribution and reduce burden on the logically centralized controller?
- Can the network architecture support real-time interactive video applications such as telesurgery that require reliability from the network?

- Can the network architecture support multi-domain end-to-end path selection?
- What is the message complexity of network architecture which supports feasible path selection for real-time interactive video applications?

  These fundamental questions help guide the development of network architecture that meets the needs of real-time interactive video applications such as videoconferencing and distance learning.

1.5 Assumptions and Limitations

This experimental thesis research uses a network simulator to evaluate the propose network architecture. The network architecture is a stable version of the network simulator. Regression testing and code reuse [14, 15] allow the effect of bugs in simulation to be minimized. This thesis work does not include deployment of the network architecture into production environment—testbed; therefore, a certain level of trust is needed when working with simulations [14, 15].

1.6 Expected Outcome

This thesis involves developing a network architecture that selects a feasible QoS path among multiple paths for real-time interactive video applications. This thesis expects to develop a prototype that illustrates and captures the architectural elements and behavior of the proposed network architecture. This thesis expects to focus on the message complexity of network architecture and how state distribution affects scalability of the network architecture. This thesis focuses on API usability—design API to accept minimal input from application developer. This thesis research blends software engineering, distributed systems, and computer networks to developed a network architecture that provides QoS for real-time interactive video applications.

1.7 Scope

This thesis focuses on developing a network architecture that supports selecting a feasible end-to-end QoS path among multiple paths for real-time interactive video applications. This thesis identifies the architectural elements and determines the message complexity of the proposed network architecture. Although dummy packets were transmitted through network, this thesis is not concern with content of the packets or how video encoders and hardware plays video content. The proposed network architecture is agnostic to network vendors. Justifying the proposed network architecture cost benefits is outside the scope of this thesis.

1.8 Dissertation Structure

- Chapter 2 Software-Defined Networking Survey: A Research Landscape surveys state-of-the-art in SDN and presents the key SDN architecture concepts. Chapter 2 presents a taxonomy for classifying related works and illustrating where this thesis fits into SDN research.

- Chapter 3 Video over Software-Defined Networking (VSDN) presents network architecture that selects a feasible path using the network-wide view. Chapter 3 describes how video application developers use protocol for requesting network service. Chapter 3 presents the results of implementing VSDN prototype and evaluates performance of VSDN.

- Chapter 4 Explicit Routing in Software-Defined Networking (ERSDN): Addressing Controller Scalability presents routing scheme that selects transit routers at the network edge. Chapter 4 presents the design and implementation of ERSDN. Chapter 4 evaluates the effect of ERSDN on the scalability of SDN controller.

- Chapter 5 Reliable Video over Software-Defined Networking (RVSDN) builds on Chapter 3 and addresses the issue of finding the most reliable path. Chapter 5

presents the design and implementation of RVSDN. Chapter 5 presents the results of implementing the RVSDN prototype and evaluates the behavior of RVSDN.

- Chapter 6 Multi-Domain over Software-Defined Networking (MDVSDN) presents network architecture that selects end-to-end QoS path for real-time video applications across independent domains. Chapter 6 describes the architectural elements of MDVSDN. Chapter 6 presents the results of implementing MDVSDN prototype and evaluates the behavior of MDVSDN.

- Chapter 7 Conclusions summarize thesis research, highlight thesis contributions, and suggest where thesis results can lead in the future.

2 SOFTWARE-DEFINED NETWORKING SURVEY: A RESEARCH LANDSCAPE

2.1 Abstract

Virtualization of compute and storage resources creates a more flexible and manageable infrastructure. The computer network is unable to meet the demands of infrastructure that supports network services such as compute, storage, and security and network applications such as quality of service, traffic engineering, and load balancing. The infrastructure demands require the computer network to install network flows and inform network services and applications about network state changes, allowing the services and applications to behave intelligently—react and respond to the network state changes. Traditional networks are too complex for supporting services and applications in a seamless, efficient, and cost effective way. Software-defined networking has been adopted as the future network architecture for addressing the need for rapid, deployable, and dynamic network services while giving network applications and services control over the behavior of network.

This chapter surveys the state-of-the-art in software-defined networking and presents the key concepts of software-defined networking architecture. This chapter reviews past programmable networking research that ideas impacted the development of software-defined networking. This chapter summarizes a landscape of software-defined networking research and identifies the key software-defined networking contributions of research. This chapter identifies common set of characteristics among research and creates taxonomy for better understanding of field. This chapter highlights and discusses future software-defined networking challenges. This chapter discusses the network industry software-defined networking state. This chapter identifies the key in-

novations pushing the software-defined networking paradigm shift and draws together the contents of chapter.

2.2 Introduction

The Internet Protocol (IP) network architecture is unable to satisfy the demands of network applications which impair network innovation. The network equipment vendors, developing hardware and software, release cycles are long, slowing innovation of network applications and services. Researchers believe developing network hardware and software separately speeds up innovation of the network applications [16], reshaping relationship between the network and applications.

The computer network is made up of three planes—the data plane, control plane, and management plane. The data or forwarding plane is responsible for forwarding a packet out network port. The control or decision plane is responsible for computing route of a packet. The management plane provides an interface to the computer network, allowing network operator to configure and manage network resources. The control plane and data plane of network devices are tightly coupled, making the network devices ill-suited for meeting requirements of enterprise and carrier networks [16].

Software-defined networking (SDN) [17–20] proposes decoupling of the data plane and control plane to enhance application innovation [16]. In SDN, the control plane is logically centralized in a server—SDN controller or controller. The data plane is implemented in commodity network equipment—switch. The data plane forwarding tables or flow tables are controlled by the SDN controller which hosts external control processes—network applications and network services.

The network applications such as traffic engineering and load balancing program the flow table of switch, giving network operator control of the network —behavior. The network services program the flow table of switch, creating innovative services such as compute, storage, and security.

The controller communicates with switch over an open interface using OpenFlow [21]. OpenFlow is a standard protocol that remotely controls the data-path of a switch. OpenFlow enables the flow table of a switch to be programmed by network applications and network services running on the controller. Programming of flow table constitutes forwarding actions that cause a packet to be dropped, forwarded to a port, or forwarded to the controller.

SDN promotes rapid development and rapid deployment of network applications and network services [22]. SDN enables network policy driven end-to-end quality of service (QoS). SDN allows network resources to be optimized and automated, resulting in cost saving [16, 23]. The benefits of SDN have been realized in campus networks, wireless networks, and data center networks.

HP [24], NEC [25], and Big Switch [26] are developing OpenFlow enabled switches. Network equipment vendors such as Arista [27], Ericsson [28,29], HP [24], Juniper [30], NEC [25], and IBM [31] are developing SDN controllers. The absence of SDN controller Application Programming Interface (API) standard has caused network vendors such as Extreme Networks [32] and IBM [31] to developed the complete SDN stack—OpenFlow switch and SDN controller, avoiding interoperability issues.

The research community sees SDN as a critical part of designing a more flexible and optimizable, cost effective [23], and scalable computer network [16]. The research community believes SDN rapidly enhances application innovation. There are open issues, challenges, and obstacles for the research community to explore and overcome before the impact and value of SDN are understood.

This chapter surveys the state-of-the-art in Software-Defined Networking (SDN), discusses past programmable networks research, gives an overview of SDN architecture, reviews SDN research, presents a taxonomy for understanding of SDN research, discusses SDN in networking industry, and identifies future research challenges. This chapter identifies the key innovations that are pushing SDN.

The remainder of this chapter is organized as. Section 2.2.1 discusses programmable networks background. Section 2.3 gives an overview of SDN architecture. Section 2.4

discusses the key contribution of SDN research using a set of common characteristics. Section 2.5 discusses research challenges. Section 2.6 discusses the state of SDN in networking industry. Section 2.7 summarizes the key innovations that are pushing SDN and draws together the chapter contents.

2.2.1 Programmable Networks: Background

In the past, the network software and hardware have been tightly coupled, making it difficult to rapidly deploy network services. Past programmable network research efforts have increased the ability to deploy network services, evolving over time. SDN is an evolution of programmable network research ideas [33] which aims were to increase programmability of the network and allow rapid deployment of network services and applications.

This section discusses past programmable network research that ideas have led to evolution of the network—SDN.

Tutorial on Intelligent Networks [34] builds an intelligent network (IN) that allows flexible routing and metering, advanced user interaction, and advanced user control. IN allows rapid deployment of standard vendor agnostic network services by separating network services from the network. The network services runs within service control point (SCP) that are responsible for querying service data point (SDP), a database that stores user data. The calling card service and universal access number service are services deployed using IN.

Active Networks [35] (AN) are programmable networks that respond to mobile code and mobile data encapsulated in packets. There are two approaches for programming AN. The first approach is discrete approach used by programmable switch, acting on packets that contained small programs where the packet header determines program execution. The second approach is an integrated approach where messages are used for encapsulating small programs—capsules. The switch receives capsule and acts on the capsule and sends the capsule to destination. AN allows the mobility

of programs between switches. AN allows protocols to be deployed easier, enhancing application innovation. Network applications such as firewall and web proxy are deployed using AN.

Cabletron's SecureFast VLAN Operational Model [36] implements distributed connection-oriented switching protocol that provides layer-2 forwarding. The connections are programmed into switch. The connection mappings between input port and output port allow programmatic control over packet routes. A switch queries another switch connection mapping, allowing the switch to make intelligent decisions.

General Switch Management Protocol (GSMP) [37] performs efficient and cost effective handover in Multiprotocol Label Switching (MPLS) networks. GSMP have slave switches and a master controller that establishes and releases network connections. GSMP introduces the idea of having programmable switches programmed by a centralized network controller.

IP Multimedia System (IMS) as Next Generation Network (NGN) Service Delivery Platform [38] allows decoupling of call control from network applications, supporting rapid deployment of multimedia services. IMS combines voice service and packet service into single service. IMS allows service providers to distinguish themselves from their competition by rapidly deploying network services such as Voice Call Continuity.

Routing Control Platform (RCP), a logically centralized platform, addresses scalability issues of full mesh network topologies [39]. RCP communicates routes between routers and mitigates route reflection issues such as protocol oscillations and persistent loops. RCP collects information about external network destinations and internal network topology, using the information for selecting routes between routers. RCP performs routing decisions in large networks.

4D increases the manageability of network and allows network changes to be made without breaking functionality [40]. 4D investigates relationships between configuration errors of network, size of router configuration file, and routers that are manually configured by the network operator. To reduce complexity of control and management planes, 4D purposes refactoring the network using three principles—expressible

network-level objectives, global network view decisions, and network behavior which is directly controlled by the network operator [40]. The three principles of 4D imply a clean slate for designing the network architecture [40], a precursor to SDN.

SANE [41] addresses the network security issues caused by complexity of applying routing policies and bridging policies with other mechanisms such as access control lists (ACL) and middleboxes. SANE uses centralized controller for issuing capabilities which are encrypted network client source routes. The capabilities are verified at each router in path. SANE achieves better protection for enterprise networks using a centralized controller for managing network security [41].

Ethane [42] addresses complex management and configuration of the network. Ethane builds on the research of 4D architecture [40]. Ethane uses centralized controller to perform network configuration of single flow-based Ethernet switch [2]. The network client requests are sent to the controller where network policies are applied consistently. Ethane was first applied to campus networks [42].

SEATTLE [43] reduces network complexity of large enterprise networks. SEATTLE, layer-2 architecture, addresses the scalability limitation of Ethernet—poor scaling caused by broadcasting and inefficient path selection of the spanning tree protocol. SEATTLE uses centralized controller to locate network client and prevents broadcasting to find client when the client location is unknown. SEATTLE uses switch level link-state discovery protocol to compute shortest path. SEATTLE routing protocol is a self-configuring protocol, avoiding manual configuration of addressing and subnetting configurations [43].

SDN builds on the previous programmable network efforts for creating a flexible, loosely coupled, and manageable network architecture. The question asked is—why SDN now? The timing for new ideas is paramount [44]. There were no video, cloud computing, virtual machines, virtual machine migration, smart mobile devices, and large scale data centers during the earlier research; these technologies have placed great demands on the network [45,46], requiring the network to be built to match the needs of network applications and services [47].

2.3 SDN Architecture

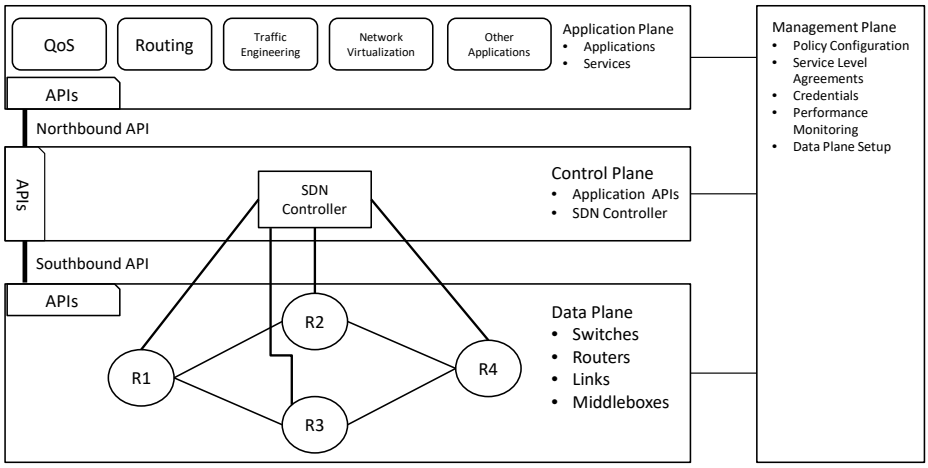

Figure 2.1.: SDN architecture, illustrating relationship between network planes. Each network plane represents specific function of the SDN architecture. In the application plane, the network applications program data plane using northbound API. Using southbound API to send requests, the control plane converts the application plane requests to modifications or queries in the data plane. The data plane presents the control plane with an open API. The management plane performs management functionalities such as configuring network policies, monitoring network performance, and setting up network devices in the data plane.

The management plane is an interface to the data plane, allowing network operator to setup of network devices. The management plane interfaces with control plane and allows the control plane to perform performance monitoring of the network such as multi-tenant networks [48]. The management plane provides an interface to the application plane and allows network applications to verify service level agreement (SLA)—network policy. The management plane configures network policies and manages cache performance such as miss rates and object sizes, allowing network operator to adjust protocol-specific parameters to optimize the network [49].

The application plane includes network applications such as performance monitoring, traffic engineering (TE), and QoS and network services such as routing, service chaining, and wide area network (WAN) optimization. The network applications optimize business processes and work-flows, automating creation of the network. For example, finding the Constrained Shortest Path First (CSPF) can be performed because the TE application has a global network view—a view of the complete network topology.

The control plane contains the SDN controller or controller. The controller presents an abstract view or graph of the state of data plane to network applications. SDN applications can operate on the abstract view of the state of network [50]. The controller runs a network operating system which uses virtualization to hide and decouple the application plane from data plane. The control plane enforces network policies which are set by the network operator. The network policies are applied consistently by the control plane which has the global view of network. The control plane includes a special controller such as FlowVisor [51] which orchestrates interactions among controllers and switches.

The data plane is composed of routers, switches, links, and middleboxes. The data plane includes flow table, Ternary Content-Addressable Memory (TCAM), and network device counters [52]. The data plane allows programmatic access to the flow tables of network resource using an application programming interface (API) such as Representational State Transfer (REST), vendor specific, or OpenFlow [21]. The data plane performs statistics gathering which is used by the control plane and application plane when making decisions.

SDN is a network architecture that separates the data plane and control plane. As shown in Figure 2.1, the SDN architecture has four network planes or functional planes. The control plane is logically centralized in the SDN controller [53]. The SDN layered architecture allows network services and applications to be separated from the data plane, improving ability to debug and troubleshoot the network [54]. In Figure 2.1, the application plane—network applications runs on network operat-

ing system [55] residing on the controller. The network applications are deployed independently of the devices in data plane. The management plane performs configuration and management functionalities such as setting up network devices in the data plane [53]. The network applications, such as TE and routing, have control over how network traffic flows through network. The SDN architecture increases application awareness [56–59] by allowing network applications to decide when and how to act upon network related events. For example, the routing application can program the data plane and send network traffic through path R1-R3-R4 after a link failure—link-R1-R2, in Figure 2.1.

2.3.1 Communication Between Network Planes

In SDN, communication between network planes is performed over an open interface—API. The APIs that communicate between planes are organized by their functionality. In Figure 2.1, there are three APIs—the northbound, southbound, and east-westbound [60, 61] which is not shown.

The application plane and control plane use the northbound API to communicate with each other. The control plane and data plane use the southbound API for communicating with each other. A federation of controllers uses the east-westbound API [60, 61] to communicate with one another.

The network applications or services make configuration changes to the data plane using northbound API. The network applications send high-level modifications or query requests to the controller.

For example, in Figure 2.1, the network application can be a web browser that the network operator uses to configure data plane. In the web browser, the network operator is presented with single node graph with outgoing links. The network operator configures the QoS such as bandwidth, delay, and jitter of outgoing links. To configure QoS between two outgoing links—ingress and egress, the network operator connects the two links and sets the QoS values. After the network operator saves con-

figuration changes, the changes are sent to the control plane or controller over a secure connection such as HTTP over SSL where high-level requests from the application plane configures data plane—physical links.

The graph that is presented to network operator hides the details of data plane —R1, R2, R3, and R4, in Figure 2.1. The network operator connects link-A that represents R1 and link-B that represents R4 to each other and configures the QoS values between R1 and R4, in Figure 2.1. The network operator sets the QoS values on single logical link—link-A-B which in the data plane is represented by two physical links—R1-R2 and R2-R4. The control plane maps the single logical link—link-A-B in the application plane to two physical links in data plane.

The control plane determines the single logical link A-B maps to physical links R1-R2 and R2-R4—path R1-R2-R4. Using the southbound API, the control plane sends a port modification requests to R1, R2, and R4, creating path R1-R2-R4. The switches send status messages to the control plane using southbound API.

The controller uses the northbound API for sending configuration response from data plane to application plane. The response is presented to network operator as successful configuration—link-A-B is shown with QoS values set as specified.

The high-level configuration and modification requests from the application plane travels downward through the SDN planes and are converted to particular configurations and modifications in the data plane. The data plane state changes travel upward through the SDN planes to control plane. The state changes are forwarded from the control plane to application plane—network applications and services.

2.4 SDN Research Review

This section reviews SDN research and identify the key contributions of research. The SDN research are organized and classified using common set of characteristics that were discovered while reviewing the research.

The SDN research identified during the synthesis of our research are summarized in Table 2.1 and Table 2.2.

2.4.1 Characteristics

This chapter uses four common characteristics for better understanding SDN and how SDN research are related.

- *Network technology* determines programmability of SDN. The network technology such as data center networks, wireless networks, and home networks determines amount of programmability. For example, the wireless networks programmability is decreased because network resources such as CPU, memory, and bandwidth are limited. Data center networks have a plethora of network resources which need complex management schemes and policies.

- *Layer of control* refers to network plane which network operator is able to control the behavior of the network. For example, the network operator is able to control the behavior of data plane using REST API which provides advanced switch configuration options.

- *Application domain* indicates the functionality of SDN applications such as routing, load balancing, or network management [62]. The design choice that delivers the SDN application is constrained by the application domain. For example, a traffic engineering application requires the network-wide or global view. Partitioning the network over multiple controllers affects the design of traffic engineering application [63, 64].

- *Level of programmability* indicates network plane which network service is introduced. Level of programmability is coupled to the API exposed by each network plane. Introducing service at the management plane requires communication with the network using command-line interface (CLI) [65]. Introducing service

in the control plane requires communication with an open programmable interface. Introducing service in data plane requires communication with the network using OpenFlow [21] or General Switch Management Protocol (GMSP) [66] such as Forwarding and Control Element Separation (ForCES) [67].

2.4.2 Network Technology

This section discusses and organizes SDN research using the network technology characteristic such as the Internet and wireless networks.

Internet

Software-Defined Internet Architecture (SDIA) [68] decouples Internet Protocol (IP) of the Internet from infrastructure, routers, switches, and links, changing how the Internet is built. The IP is embedded in applications and routing protocols of the Internet, making changes for the Internet difficult [68]. OpenFlow based solution for the Internet are too specific [68, 69]. SDIA [68] provides a generalized solution, whereas OpenFlow networks [70] solution is specific. A more general solution allows packets to be routed between autonomous systems (ASes) using standard technologies such as middleboxes, Multiprotocol Label Switching (MPLS), SDN, and software-based forwarding [68]. SDIA services are built into software on the edge of network above layer-2 and the network core performs layer-2 services only, allowing services to be attached to the network without major changes.

The solution presented in [71] improves inter-domain routing of the Internet by developing a backwards compatible routing model which uses outsourcing [72]. The solution [71] addresses deficiencies such as scalability, security, and complexity of the Internet routing protocol—Border Gateway Protocol (BGP). Enterprise businesses increase routing efficiency and prevent policy conflicts by outsourcing their routing decision to service providers. Outsourcing routing simplifies enterprise networks by

removing routing protocols and allowing experts to optimize routing which improves network performance and increases network security.

Cloud and Data Center Networks

SEATTLE [73] reduces packet flooding and broadcasting of Ethernet—Address Resolution Protocol (ARP) and Dynamic Host Configuration Protocol (DHCP). SEATTLE captures broadcast packets and converts the broadcast packets into unicast transmission that is delivered directly to destination. SEATTLE uses a centralized consistent hash lookup table to locate the destination. The lookup table resolves the Media Access Control (MAC) address of destination to IP addresses and physical location (Top of Rack (TOR) and End of Row (EOR)) of the destination to MAC address. SEATTLE supports virtual machine (VM) migration using a caching scheme.

Portland [74] makes layer-2 (L2) switching more efficient [75, 76] by using location discovery protocol (LDP) which allows switch to determine its location in the network. LDP communicates the switch location in network to another switch. Portland uses pseudo MAC address (PMAC) that encodes the location of network client or client in the network—multi-rooted tree instead of using the MAC address of client, allowing packet forwarding using the PMAC prefixes instead of the MAC address. Portland uses centralized fabric manager for resolving ARP queries, simplifying multi-cast, and enhancing fault tolerance. Portland separates the client identity from client location, enabling unmodified client to freely move around and be located in data center.

Virtual Layer 2 (VL2) [77] provides a virtual service which creates the illusion of having a single switch for entire network. VL2 removes network bottleneck for client-to-client communication—uniform communication. VL2 allows services to run in isolation without affecting the quality of one another. VL2 addresses client mobility by modifying the client and remaining backwards compatible with legacy network technologies such as Equal-Cost Multi-Path (ECMP) routing, Label Switching Router

(LSR), and Internet Protocol (IP) multi-cast. In VL2, a network service has an application specific IP address (AA) and its own virtual subnet. VL2 maps AA(s) to location specific addresses (LAs) that describes the location of service in CLOS network. The centralized lookup service runs on the client instead of in the network.

Two data center design objectives are to increase scalability and flexibility of network by performing network optimization such as eliminating layer-2 broadcast and exposing the possible network paths. Wide-area Layer-2 (WL2) [78] achieves scalability similar to SEATTLE by rerouting control plane traffic to the controller, aggregating layer-2 routing, and creating fast flow setup overlay. WL2 [79] and Hierarchical SDN (HSDN) [80] expose the possible paths in the network, increasing scalability and flexibility of the network. WL2 [79] uses source routing while HSDN [80] uses hierarchical underlay partitioning, taking full advantage of physical topology of data center to improve routing scheme.

M2cloud [81] framework provides scalable network control for multi-site data centers. M2cloud [81] provides inter data center traffic optimization and cross-site performance isolation for tenants. M2cloud uses two level [82] controller architecture where the local controller performs flow table configuration and global controller performs inter data center traffic engineering and global workload balancing [81]. Data centers are connected by programmable border gateways that are controlled by the global controller, allowing load balancing to be performed among data centers. M2cloud improves inter data center bandwidth utilization [83] 24% using the optimal path to send the traffic of tenants between data centers.

SWAN [83] improves inter data center bandwidth utilization by controlling how much traffic a network service sends and re-configuring inter data center traffic paths to match demands of the network. SWAN performs link re-configuration in a congestion free manner by leveraging small amount of link scratch capacity and forwarding table memory. SWAN minimizes forwarding rules by dynamically changing with the traffic demands or available paths in the network. SWAN carries 60% more inter data center traffic than MPLS-TE.

Wireless Networks

The authors [84] propose to use controller applications to simplify the design and management of cellular networks. The solution [84] uses SDN to express high level policies on subscriber basis and apply real-time control to the network traffic using switch agents. The solution [84] uses SDN for performing deep packet inspection (DPI) [85, 86] and header compression on packets and for managing the resources of base-station remotely.

Odin [87] allows enterprise Wireless Local Area Networks (WLAN) to be programmed by network operator. Odin simplifies client management through use of virtual Access Point (AP). The virtual AP allows each client to be isolated from one another. Odin performs client hand-off between AP in software, using agents running on the AP to communicate with the Odin service running on Odin controller. Applications such as seamless mobility, smooth hand-offs, load balancing, and mitigating hidden node problem are built on top of Odin [88]. OpenSDWN [89] extends Odin and introduces service differentiation using per-flow WiFi datapath transmission rules.

OpenRadio [90] is a wireless programmable data plane which provides modular and declarative programming interface across the wireless protocol stack. OpenRadio separates the wireless protocol into processing plane and decision plane, enabling network operator and network provider to remotely program base-station. OpenRadio reduces the manual upgrading time of base station and allows programming of the network infrastructure to be performed in software—network optimization.

SoftRAN [91] is a software-defined centralized control plane for radio access networks which reduces complexity of wireless networks. SoftRAN abstracts the control plane from each local geographical area into virtual big base station. The virtual big base station is comprised of the central Radio Access Network (RAN) controller that makes decision regarding hand-overs and interference management and an individual physical base station which contains minimal control logic. A single SoftRAN [91] controller manages base stations in a geographical location.

OpenRAN [92] is architecture for software-defined RAN. Similar to SoftRAN [91], OpenRAN [92] addresses heterogeneous interconnected characteristics of wireless networks. OpenRAN [92] achieves complete virtualization and programmability, making RAN open, controllable, flexible, and evolvable. OpenRAN [92] abstracts and combines control functions of RAN [88] and places them in the controller. The OpenRAN [92] controller creates and optimizes virtual access elements spectrum allocation and compute and storage resources.

ProCel [93] is a cellular network architecture that eliminates unnecessary processing of flows in the Long-Term Evolution (LTE) core network. ProCel architecture has two classes of services—non-real-time and real-time. The non-real-time traffic is routed directly to the fixed IP network instead of LTE core. The real-time traffic that requires strict QoS is routed through the cellular network [93]. ProCel reduces data traffic and control traffic at the mobile core network, reduces network latency, enhances application innovation at the edge of network, and enables service providers to support large class of applications without deploying complex backhaul and core networks [93].

Carrier Networks

The Information and Communication Technology (ICT) project Split Architecture (SPARC) is a network architecture developed for integrating SDN into carrier networks [94]. SPARC [94] is driven by the growth and influence of the Internet that lacks security, scalability, and mobility. The carrier networks are complex and diverse with multiple technologies and protocols that were developed for addressing specific network problems. The carrier networks are using proprietary software and hardware, causing solutions to vary between vendors. SPARC outlines required design changes to integrate SDN into the Internet. SPARC uses a hierarchical controller in the control plane and splits forwarding and decision elements in the data plane.

The authors of [95] investigate the ability of SDN to provide carrier grade functionalities in areas of reliability and energy efficiency [96] and summarize requirements for carrier grade resiliency in the data and control planes. The authors [95] reduce network power consumption to improve network scalability and reduce the carbon footprint. The solution [95] uses multilayer traffic engineering to conserve energy and reduce the carbon footprint of network. The OpenFlow switch requires modifications to support power management because power management was excluded from the original switch design [95]. The forwarding engine and power supply are the greatest energy consumption components [97] of the switch.

Home Networks

The solution [98] reduces the complexity of managing and configuring home networks. The solution [98] gives the users visibility into their home network—network performance and network policies that are applied to their network. The solution [98] improves the user experience by changing the way home network users interact with their networks.

The home network users want an intentional user interface that directly maps user intentions to the configured network policies [98]. The solution [98] creates an outsourcing business model [72, 89] home network configuration tasks to service providers. Outsourcing home network configuration relieves user of complex task of configuring and managing the home network [72, 98].

The solution presented in [99] reduces cost and increases manageability of the home network. The solution [99] uses slicing [100], a concept of having two or more virtual networks share the same physical infrastructure to increase manageability of home networks.

Slicing allows multiple service providers such as Internet, cable TV, and electrical power to share the home network without interfering with one another. Slicing

allows network providers to share cost, reducing the overall cost of operating home network [99].

There are multiple standards used for configuring home network devices which increases network manageability. The solution [101] uses cloud SDN and OpenFlow for auto-configuring home networks. The solution [101] allows home network auto-configuration and management without requiring specific standards equipment or middleware. The solution [101] uses a home database that stores the MAC address of network device. The MAC address identities the manufacturer of the device [101] which allows the network devices to be automatically detected when connected to the network.

Enterprise Networks

Floodlight [102], formally Beacon [103], is an OpenFlow controller developed by Big Switch Networks [26]. Floodlight is an enterprise class controller with a collection of built-in network applications—circuit pusher, OpenStack Quantum Plugin, packet forwarding application, and firewall application. The Floodlight controller implements common OpenFlow network functionalities, such as link discovery, flow cache [104], tracing, and monitoring [105, 106]. Floodlight network applications increase controllability and manageability of enterprise networks. The Floodlight controller supports three APIs—REST, module, and OpenStack. Floodlight dynamically establishes a sequence of virtual middlebox functions that supports adaptive network service chaining [107].

RouteFlow Control Platform (RFCP) [108] builds on idea of Routing Control Platform [39]. RFCP [108] uses centralized controller hybrid networking model to provide a global network Border Gateway Protocol (BGP) routing service. RFCP hybrid model supports integration of traditional and OpenFlow network devices [109]. RFCP addresses network deficiencies such as routing loops, protocol oscillations, and unoptimized path selection that are caused by the complexity of control and management

planes. RFCP provides fine grain control over network resources, and more flexible and intelligent based routing. RFCP addresses four network issues—centralized BGP [110], OpenFlow data path performance, need for high availability [111–113], and switch flow table limitation.

Configuring RFCP is a manual process that takes days for network operator to complete; therefore, the authors [114] developed a framework that discovers the network configuration and automatically configures RouteFlow using configuration messages [114].

Campus Networks

Stanford University partnered with 7 other universities to start the Clean Slate Program [115, 116] in 2009. The Clean Slate Program (CSP) supports GENI [100] which is a virtual laboratory for exploring scalable Internet innovations. CSP researchers believe the collaboration of campuses increases innovation for researchers and network administrators.

There are multiple active campuses collaborating on CSP [115, 116] and sharing challenges and successes in deploying SDN. The CSP strategy for expansion of campus deployments is to encourage SDN in marketplace, train engineers in SDN using classes and workshops, provide continuous support to trained engineers, and help with research funding [115, 116].

2.4.3 Layer of Control

This section discusses and organizes research that contribute to characteristic of layer of control.

Data Plane

Open vSwitch [117] started from collaboration between Nicira Networks and the University of California, Berkeley. vSwitch runs in software, but unload router processes to hardware. vSwitch address virtualization related issues—VM mobility, network scalability, VM isolation, and traffic isolation [118]. vSwitch allows VM configuration state, Access Control List (ACL), QoS policy, and Layer-2 (L2) soft states to be migrated. vSwitch allows fine grain control and programmability of the forwarding table of switch and supports tunneling, firewalling, and filtering.

Open Transport Switch (OTS) [119], a virtual OpenFlow switch architecture, reduces the complexity of control plane in optical transport networks [120–125]. OTS [119] controls packet-optical cross-connect (XCON) and bandwidth allocation capability of the optical switch. OTS allows QoS aware applications to request provisioning of circuits cross-connects or aggregation of packet interfaces into optical trunks with required QoS [126] metrics such as bandwidth. OTS setup time of an end-to-end circuit is similar to setup time of distributed network algorithms, a promising discover for service providers [119].

NetFPGA [127] is a programmable line-rate, flexible, and open source platform that allows universities and researchers to prototype network hardware for researching and teaching. NetFPGA removes the need to rely on network equipment vendors for features, opening network innovation to a broader group of researchers. NetFPGA has hardware limitations [128], but can run a complete network on 4-port NetFPGA [129].

SwitchBlade [130] is a packet processing pipeline platform that allows customized protocols to be rapidly deployed. SwitchBlade builds on the concept of Click [131]. SwitchBlade maintains same flexibility and configuration strengths of Click and addresses performance issues of Click, providing high performance packet switching. SwitchBlade balances programmability of software and performance of hardware, enabling rapid network prototyping and protocol deployment. SwitchBlade data planes are customized and run in isolation on the same NetFPGA [128].

DIstributed Flow Architecture for Networked Enterprises (DIFANE) [132] is a switch architecture that scales by efficiently keeping traffic in the data plane. DIFANE selects and directs packets through intermediate switches or authority switches. DIFANE [132] addresses the scalability issue of OpenFlow that is caused by excessive control plane events—packets. The authority switches perform similar functionality as the controller, minimizing the packets sent to control plane and increasing network throughput. DIFANE partitions flow rules across authority switches, avoiding having the controller store rules [133]. Each rule is mapped to a single authority switch.

DevoFlow [134] decreases full visibility of the controller over network events—packets, reducing the interactions between the control plane and data plane. DevoFlow reduces the overhead in control plane by devolving the functionality of switch that is displaced to the controller and places the functionality in the switch. The controller maintains control over significant flows—QoS sensitive, but normal flows are processed in the switch using wildcard rules. The wildcard rules aggregate multiple rules into one, reducing interactions between the controller and switches and flow rules installed in switch. DevoFlow builds on idea of [135], performing rule based cloning in the data plane instead of control plane.

Consolidated Middlebox (CoMb) [136] is a top-down design for a middlebox infrastructure that addresses difficulty of provisioning and managing [137] middlebox technologies. CoMb consolidates middlebox network services into a centralized controller where middlebox applications share the same physical infrastructure. CoMb supports network service chaining [107] and allows middlebox functionality to be mobile [138].

The authors [138] developed a network management system that allows dynamic instantiation and quick movement of middleboxes. The solution [138] supports a fully programmable network that provisions middleboxes on-the-fly—as needed [139].

The authors [140] simplify traffic steering through middleboxes. The solution [140] allows network operators to specify a logical middlebox routing policy and auto-

matically translates the policy into flow rules that are optimized using the physical topology, switch capabilities, and resource constraints of middlebox.

Application-ware data plane processing [57] in a middlebox [59] provides network functions from layer-4 through layer-7, a requirement as SDN is deployed and integrated into the traditional network infrastructure. Software-Defined Middlebox Networking (SDMBN) [141], a middlebox framework, simplifies management of complex middlebox deployments by supporting dynamic middlebox control scenarios.

Research that increase programmability of network devices have emerged to overcome the lack of programmability and performance of traditional network devices. Reusable abstractions have been developed for supporting special processing in data plane—switches [109, 142–144] and middleboxes [145]. Soft switches such as EdgePlex [146] and mSwitch [147] increase data plane flexibility, reliability, and throughput.

Control Plane

Onix, a distributed controller, runs on multiple servers. Onix gives network operator one view of the data plane and provides a common control management platform, increasing manageability of network [148]. Handling state distribution [149] and collecting information about the control plane are functions of the Onix controller [148]. The Onix controller stores network state in the Network Information Base (NIB) which enables network applications to register for state notifications. The network state is partitioned and replicated among distributed controllers which provide durability and consistency [150] while improving scalability of the control plane.

HyperFlow [151] is a logically centralized and physically distributed controller. The designers of HyperFlow [151] identify and address three scalability issues with a single controller: the amount of control traffic generated by switch, delay between switch and controller which is caused by controller geographical displacement—controller placement problem [152–156], and processing bottleneck of a single

controller. HyperFlow scales horizontally which gives network operator the ability to deploy controllers as required.

Kandoo is a configurable and scalable control plane [82] that uses two layer controller architecture—bottom controller and root controller. The bottom controller or local controller maintains its local network state and controls its own switch. The top controller or root controller maintains the global network state, giving the root controller visibility of entire network. The local controller processes frequently occurring events from switch, reducing the events processed by root controller. The root controller runs non-local control applications and delegates flow installation to the local controller. Kandoo is designed for applications that use the local network state such as link layer discover protocol and local policy enforcer [82]. Applications such as routing [110] require the global network state and are unable to be offloaded to local processing [82].

Dynamically Reconfigurable Processor (DRP) [157] uses an on chip routing diorama for improving scalability of controller. DRP performs routing processing on a chipset before pushing modifications to the data plane. The performance of network services such as finding Constrained Shortest Path First (CSPF) improves using DRP network emulation in hardware. DRP uses parallel graph partitioning to speed up finding the CSPF. DRP is used for traffic engineering and routing [157].

Maestro is a multi-threaded controller that was developed at Rice University [158]. Maestro exploits parallelism of architecture—multi-core processors of servers to improve performance of the controller. Maestro addresses the scalability issues of the NOX [55] controller which uses a single thread to process network events. Maestro maintains single threaded programming model for application developer, but internally provides parallelism to increase throughput. Maestro distributes the work evenly among processor cores which reduces cross-core communication overhead and minimizes per-flow memory usage.

An Open Framework for OpenFlow Switch Evaluation (OFLOPS) [159] provides detailed performance measurement of OpenFlow switch implementations [159–161].

OFLOPS explored unexplored performance measurement of the OpenFlow switch. OFLOPS tests the capabilities and bottlenecks between the application plane and data plane. OFLOPS gathers switch data using hardware and software instrumentation. OFLOPS measures five capabilities of OpenFlow switch implementations—flow table consistency, flow setup latency [162], flow space granularity, packet modification capability, and traffic monitoring capability.

NOX-MT [163] is a multi-threaded controller that establishes a lower bound on controller response time and throughput. NOX-MT uses optimization techniques such as IO batching and threading to increase the performance of controller. NOX-MT achieves optimized response time and throughput compared to other controllers [103, 158]. NOX-MT processes 1.6 million requests per second (rps) which is short of the 10 million rps required for data centers.

FlowVisor [51] is a special purpose OpenFlow controller —hypervisor which enables network virtualization—slices [164] among controllers and OpenFlow switches. FlowVisor shares the physical network among virtual networks and operates between the control plane and data plane of controllers, allowing FlowVisor to monitor the network events. The ability of FlowVisor to monitor the network events is used by HOTSWAP for upgrading controllers in a disruptive-free and correct manner [165]. Generalized network architecture similar to FlowVisor that uses resource delegation instead of controller coordination is developed [166]. The resource delegation framework [166] explicitly exposes resource delegation abstraction—direct control over provider equipment with verifiable constraints [166].

OpenVirteX [167] functions as an OpenFlow controller proxy between the network of operator and network of tenant and provides an infrastructure on demand service. OpenVirteX [167] differs from FlowVisor [51] in its ability to create multiple software-defined networks out of a single network using network virtualization [47, 168–173], whereas FlowVisor [51] slices flow space among tenants. OpenVirteX [167] provides tenants with their own header space [174], allowing tenants to create virtual networks of arbitrary topology and custom addressing. OpenVirteX [167] and FlowVisor [51]

focus on data plane virtualization, whereas HyperFlex [168] focuses on control plane virtualization such as controller CPU and memory. HyperFlex [168] ensures isolation of the control plane among virtual SDN tenants while protecting the hypervisor from over-utilization.

Fleet [175] is a distributed controller designed for detecting malicious administrator behavior or accidental misconfiguration [176] of network by administrator. Fleet uses a controller layer collocated with switches to allow the switches to dynamically associate with an active controller and digitally verify signatures of the controller. Fleet [175] collocates controller intelligence in the data layer to avoid reconfiguration of switches—connecting to a new controller after a failure. Fleet [175] maintains network availability while under the attack of possible colluding malicious administrators whose goal is to reduce the availability of network.

2.4.4 Application Domain

This section discusses SDN application domain—applications architecture.

The application domain describes the structure and behavior of applications. This chapter organizes applications by their structure and behavior and design choices made by the designers.

Network Optimization

The authors [177] use content-based traffic engineering to optimize the network resources by observing and extracting content metadata at the network layer, optimizing delivery of content. The solution [177] adds a content management layer to the controller layer that supports traffic engineering and firewalling. The content management layer manages content names and caching policies, translates content names into routable addresses, and performs traffic engineering [64, 177, 178]. The content metadata is extracted and used to route requests to content servers. Forwarding decisions are made in the control plane using received content metadata.

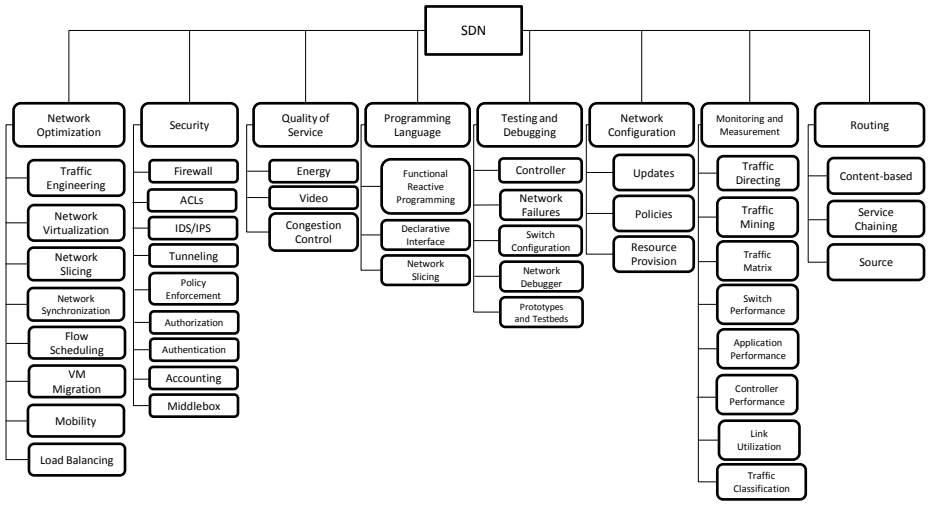

Figure 2.2.: SDN application domain, illustrating how SDN applications are organized. There are eight application research areas—network optimization, security, quality of service, programming languages, testing and debugging, network configuration, monitoring and measurement, and routing. The eight application research areas have specific applications which functionalities are researched and developed. For example, tunneling applications—tunneling supports security, and applications that update—updates the network support network configuration.

The data plane content metadata allows traffic engineering decisions to be made in the control plane.

The solution [173] supports VM migration using a network virtualization architecture. VMs are load-balanced across geographical locations to increase performance and efficiency of network applications [173]. A network application may be distributed among a collection of VMs, requiring a collection of VMs to be migrated.

LIve Migration of Ensembles (LIME) [179] allows a collection of VMs to migrate between networks. LIME clones the state of data plane to new set of switches and incrementally migrates the VMs [179]. LIME runs on the controller, allowing the required resources to be provisioned for ensemble migration.

Hedera is a dynamic and scalable flow scheduling system that uses a multi-stage data center fabric to aggregate network resources [180]. Hedera uses centralized routing with flow demand estimation and scheduling heuristic to manage the bandwidth requirements of application. Hedera increases the bisection bandwidth of network by using the global network state to schedule flows, outperforming static state-of-the-art hash based equal-cost multipath (ECMP) [181] load balancing.

Plug-n-Serve [182] is a load balancing OpenFlow application which performs traffic engineering [123, 183, 184] using the congestion [185] of network and load on servers [182]. Plug-n-Serve adds servers to network, detects changes to the network, and makes traffic adjustments that minimize server response time using the LOad-Balancing over UnStructured networks (LOBUS) algorithm [182].

OpenFlow-Based Server Load Balancing Gone Wild is a set of algorithms that exploit OpenFlow wildcard rules [186] to create a scalable solution [187]. The algorithms steer large aggregates of client traffic to server replicas, compute concise wildcard rules, and adjust to change in load balancing policies—adapting to traffic distribution [188]. OpenFlow-Based Server Load Balancing Gone Wild is as cost effective [189] as a load balancer.

Sprite [190] uses traffic engineering to improve video traffic quality. Sprite dynamically adapts the network policy to achieve high level TE objectives of the video application such as YouTube. Sprite enable video traffic to be switch between Internet Service Providers (ISP) using source network address translation to map each outbound connection from edge switch to a specific inbound ISP for return traffic [190]. The Sprite controller uses agents to gather performance metrics such as throughput and round trip delay from the switch. The metrics are aggregated and periodically fed to the Sprite controller where decision is made to swap users from one ISP to another [190].

Security

FRESCO [191] is a modular security framework that allows researchers to implement, share, and compose security detection and mitigation services. FRESCO supports chaining multiple security modules together to create a security service. For example, an intrusion detection module can be linked with a security module, producing the correct flow rules for switch. FRESCO allows session information to be shared among network applications, allowing collaboration among applications. The FRESCO framework includes interpreter, API to support security application development, and Security Enforcement Kernel (SEK) which resolves flow rules conflicts [97, 191].

FortNox [192] is a software extension to NOX [55]. FortNox resolves conflicting flow rules from network applications that share the network. FortNox intercepts and checks flow rule conflicts in real-time, using authorization roles and digitally signed flow rules. FortNox guarantees integrity among dynamic applications, a key aspect of SDN security [193].

FLOWGUARD [194] is a security framework that is similar to FortNox [192]. FLOWGUARD is designed to detect and resolve firewall policy violations [195]. FLOWGUARD [194] detects violations by examining the flow path space against authorization space in firewall, tracking flow paths through the network, and checking rule dependencies in flow tables and in firewall policy. FLOWGUARD [194] detects and resolves firewall violations dynamically when the network state or configuration changes, an issue unaddressed by previous work [192].

Quality of Service

The solution [196] provides QoS for Scalable Video Coding (SVC) encoded video. QoS flows are generated and translated into OpenFlow rules [196]. The solution [196] minimizes the route length of QoS traffic and packet loss for best effort traffic. The goal of solution [196] is to have zero packet loss for video base layer and best effort

QoS for video enhancement layer. The solution [196] uses policing at the edge network to ensure clients adhere to the service-level agreement (SLA).

The solution [197] builds on the idea of [196]. The solution [197] supports lossless reroute of the base layer for SVC encoded video. The solution [198] reduces video freezes by using the layered characteristics of SVC and dynamic routing ability of OpenFlow, optimizing video delivery during network congestion.

The solution [199] provides an SDN-based framework for supporting video IP multicast. The authors [199] use the global view of network to optimize construction and maintenance of the multicast tree between source—server serving video and video subscribers. The authors [199] consider two subscription types—standard and premium or enhanced video quality. The solution [199] provides lossless video quality over medium loaded networks for standard and premium users.

Video over Software-Defined Networking (VSDN) [200] is a QoS architecture that provides end-to-end quality of service for real-time interactive video applications such as videoconferencing and distance learning. VSDN uses the global network state to calculate video application optimal path. Video applications request network service using an API. VSDN supports three video type specifications—Common Intermediate Format (CIF), Enhanced-definition (ED), and High-definition (HD) [200].

The authors [201] introduce the concept of source-timed network flow changes, a technique that improves the QoS of packetized uncompressed video. The concept of source-timed network flow changes separates temporal inaccuracy of flow installation and precise timing of flow change using packet header changes from the flow sources [201]. The technique [201] allows uncompressed video to be switched at precise time of next SMPTE RP 168 video switching point, achieving high temporal accuracy of flow changes.

Programming Languages

Nettle [202] is a domain-specific language that builds on the principles of Functional Reactive Programming (FRP) embedded in Haskell. Nettle takes a stream of OpenFlow events and converts the events into OpenFlow messages—modifications and queries. Frenetic [203] builds on the principles of Nettle by allowing the behavior of network to be expressed using a high-level declarative programming language. Frenetic uses declarative programming language abstractions to allow the user to express the behavior of network. Frenetic [203] provides a runtime that runs on NOX [55] or Nettle controller.

NetCore is a high-level declarative programming language which expresses network behavior such as packet forwarding policies. NetCore builds on its predecessor—Frenetic [203], building on the same principles. NetCore provides an enhanced rule generating algorithms using reactive and proactive flow rule installation to maximize packet processing [204] in the data plane.

Procera [62, 205] is a declarative policy language similar to Frenetic [203]. Procera allows application to react to event stream—time of day, bandwidth, and user authentication. Procera is an expressive language, but lacks analysis. Flowlog [206] addresses the issue with Procera—arithmetic by striking a balance between expressiveness and analysis while losing expressiveness. Exodus [207] builds on Flowlog [206] by adding the ability to migrate existing network configuration to corresponding SDN controller programs.

Splendid Isolation [164] is a programming language abstraction that allows network operators to express network slices—virtual networks at a high-level. The network operator writes a separate program for each slice. Splendid Isolation [164] compiles programs and generates a global configuration that is applied to the network or sent to the translation validation tool for verification.

FatTire [208] uses a high level programming language abstraction to express fault tolerance levels and path redundancy. FatTire uses OpenFlow failover mechanism to

install rules that become active during a failure [174] condition. FatTire creates redundant paths, similar to NetCore [209], but uses a higher level programming abstraction than NetCore to express the paths. NetGen [210] is similar to FatTire because it allows the network operator to express fault tolerance and path redundancy. Unlike FatTire, NetGen only requires the network operator to specify a change to existing network configuration rather than an entirely new network configuration [210].

Maple [211] takes an algorithmic approach instead of a declarative approach to allow the network operator to express the intended behavior of network using a standard programming language to design an algorithmic policy—centralized algorithm. The authors [211] believe taking an algorithmic approach where developer thinks with structure reduces errors and redundancy and increases efficiency. Maple [211] includes highly-efficient multicore scheduler and runtime optimizer for recording reusable policy decisions.

Testing and Debugging

Mininet [212, 213] allows rapid prototyping of networks using a resource constrained computer such as a laptop. Mininet scales to hundreds of nodes using OS-level virtualization. The user creates and interacts with various networks using Mininet, scaling to over a thousand Open vSwitch [117]. Mininet enables rapid prototyping of networks, quick deployment of networks, and easy sharing of network designs among researchers.

OFRewind [74] is a testing debugging application that allows recording and replaying of network events. OFRewind reduces software errors and helps network operators find, isolate, and locate datapath limitations and configuration errors. OFRewind is a software layer between the control plane and data plane, collecting control log messages sent from the controller. The network operator performs regression testing of network applications by comparing previous control log messages and control log messages [74], identifying software errors.

VeriFlow [214] is a layer between the control plane and data plane that checks for global network invariants violations—checking on each forwarding rule insertion. VeriFlow uses an incremental algorithm [215–217] to search for potential violations of network invariants—absence of routing loops, access control policies, and virtual network isolations. VeriFlow slices the network into a set of equivalence classes using new rule and existing rules that overlap the new rule. VeriFlow pinpoints the set of packets that are affected by a network invariant violation. VeriFlow verifies global network invariants in real-time [214]. SDNRacer [218] is similar to Veriflow that detects network violations such as loss of reachability between the controller and switch.

No bugs In Controller Execution (NICE) [219] tests OpenFlow applications without modifications. NICE [219] uses model checking and symbolic execution to explore application state space. NICE reduces state space explored by pruning unnecessary transitions in application state space. The network operator specifies correctness of program and NICE verifies and explores application state space for correctness.

Ndb [220] uses breakpoints and packet backtrace primitive to identify the network sequence of events that caused network error. The network operator uses packet backtrace to examine and identify path [75, 221], and each switch action performed on a packet which help identify the issue. Ndb, a software layer between the control plane and data plane, uses a postcard or truncated packet header to collect state information about the network. The postcards are processed in the data plane to reduce impact of Ndb on the control plane.

Fs-sdn [222] addresses the problem with prototyping and evaluating accuracy of network applications before deploying to production network. Fs-sdn uses flowlets—the volume of flow emitted over a given time period instead of packets to simulate network, improving scalability of network simulator tool [212]. Fs-sdn [222] complements Mininet [212].

The authors [215] developed an assertion language to support verifying and debugging dynamic changing verification properties of SDN applications. The solution [215]

enables verification of more expressive network properties, avoiding spurious warnings. The application programmer has control over where assertions are placed in application, allowing the programmer to describe time-varying properties. The time-varying properties are related to dynamic state of the controller and are checked at various granularities to avoid erroneous transient property violations [215]. The authors [215] build on the ideas of VeriFlow [214] while adding an efficient incremental data structure.

FlowTest [223] is a data plane testing framework that systematically explores state space of data plane to verify the data plane behavior against policy goals. FlowTest models the functions of data plane such as firewall, load balancer, Network address translation (NAT), proxy, Intrusion Detection (IDS) System, and Intrusion Prevention System (IPS) as state machines where each state represents a data plane function state. FlowTest identifies the sequence of events or plan that is required for a set of data plane functions to transition from the present state to desired goal state [223].

CherryPick [224] is a scalable packet tracing technique that allows network operator to trace individual packet through the network. CherryPick [224] improves the ability to debug and troubleshoot network related issues [225–227]. CherryPick is design for minimizing data plane resource usage when tracing packets through network. CherryPick reduces flow rules of switch and packet header space required to perform packet tracing by embedding a sequence of link identifiers in packet [224]. CherryPick exploits the data center network physical topologies such as fat-tree topology which enables reconstructing of end-to-end paths [224].

Network Configuration

Participatory Networking (PANE) [228] is an OpenFlow controller that implements participatory networking. PANE allows network entities—end user, network client, and network application to participate in network management. PANE al-

lows network entities to contact network, request network resources, and provide hints about future network configuration—traffic. PANE benefits applications such as video and audio calls that benefit from future network reconfiguration.

Hierarchical Flow Tables (HFT) [229] is a hierarchical policy based framework for applying consistent policies to the network. HFT are useful where network resources are shared by multiple network entities. HFT are organizes as a tree where each node in the tree makes a decision on how a packet is processed. HFT allows policy conflicts to be resolved using the conflict resolution operator.

Software Transactional Networking (STN) [230] builds on Hierarchical Flow Tables (HFT) [229]. STN supports distributed control plane resolution for policy conflicts and serializing policy composition. STN [230] uses middleware to avoid policy inconsistencies in the data plane. The middleware takes composition of policies from multiple controllers and orders them sequentially. The network policies are committed or aborted similar to a database transaction.

Consistent Packet Processing for OpenFlow (OF.CPP), similar to STN [230], uses transactional semantics at the controller to achieve policy consistency in the data plane. OF.CPP increases packet isolation processing in the controller.

The solution [231] provides two update abstractions, install—installs a configuration on switch and wait—waits until change is made to the switch, simplifying network configuration. The solution [232] is similar to solution presented in [231] where there is a behavior when transitioning between network configurations; a large class of network properties hold between network configurations updates [232].

The authors [233] improve on the idea presented in [231], providing stronger flow consistency requirements and minimizing flow table entries. By sending the packets with configuration change to the controller, the solution [233] ensures a single configuration entry for each flow. The solution [233] installs double configurations in the flow table of switch, in worst case, but minimizes the network events in the control plane.

The solution [234] provides consistency between virtual networks configuration changes and supports consistent virtual machine migration from present network to future network. A sequence of VMs is created and migrated in order after the flow rules to install or delete [234] are determined. The network configuration solution [234] preserves bandwidth and avoid loops during VM migration.

The authors [216] develop an algorithm that performs incremental network updates in rounds. Each round inserts rules into flow table of switch until the rules are installed. The rules that controls the largest flow count are installed first which maximizes percentage of traffic affected by rule. More rounds result in less flow table space used because a smaller set rules can control a large flow count [216].

FlowTags [235] extends the SDN architecture through use of middleboxes. The middleboxes add tags to outgoing packets to allow flow based policy enforcement by the network to be performed. FlowTags minimizes changes required for middlebox vendors to support SDN. FlowTags [235] enforce policies through flow tracking which is the key contribution of FlowTags.

The solution [217] is an extended policy compiler that builds rule dependency with compilation. By using knowledge about rule dependencies, the solution [217] generates rule compact updates. The authors [217] identify two types of updates—content updates and priority updates [217]. The priority updates often dominate the size of total updates. The solution [217] eliminates the majority of priority updates which reduces the size of total updates.

The solution in [236] is similar to solution in [217]. The solution [236] reduces priority updates using an algebra that allows the hypervisor to incrementally compute correct relative priorities of new rules. The solution [236] builds on the idea of Frenetic [203] and allows the hypervisor to combine member policies in series or parallel. Using an incremental algorithm to compute rule changes, the solution presented in [236] avoids recalculating priority rules from scratch and reduces computational overhead and the size of total updates.

ESPRES [237] formulates network update problem as a scheduling problem where network updates are partitioned into a set of independent sub-updates to allow the sub-updates to be installed in parallel. ESPRES is a runtime mechanism that limits the rate of updates and reorders the updates to fully use the processing capacities of switch. ESPRES avoids overloading the switch and ESPRES uses virtual switch queues in the controller to reassess how switch commands are scheduled. ESPRES allows network updates to become functional faster [237].

The authors [238] use time-triggered network updates to achieve network consistency [227], requiring lower overhead than previous research. The solution [238] uses accurate time to trigger consistent network updates, whereas previous research used ordered or two-phase updates. The timed-triggered update yields shorter update duration than untimed update; therefore, the timed-triggered update is more scalable. Using time, SDN programmers tune the degree of update consistency [238] by making tradeoff [239,240] between TCAM memory—duplicate rules and network consistency.

Monitoring and Measurement

OpenTM [241], a traffic matrix (TM) estimator, uses routing information obtained from the controller for determining how to collect data plane statistics. OpenTM reduces query load in the data plane by intelligently selecting switches that are queried. OpenTM accurately estimates TM and converges within ten queries. OpenTM is implemented as a TM estimator on the NOX controller [55, 242].

FlowSense [243] achieves traffic measurements by performing estimation on packet and flow events, using a push-based approach. FlowSense uses control messages from switches to estimate performance, computing utilization of links between the switches. FlowSense [243] incurs zero measurement cost because it uses control traffic sent to controller, avoiding switch polling overhead [241] while maintaining reasonable accuracy.

OpenSafe [244] enables network operators to redirect network traffic to security monitoring applications at line rate. OpenSafe includes a flow specification language that simplifies the management of network appliances and OpenSafe allows network traffic to be monitored efficiently. LiteFlow [245] is similar to OpenSafe, but route traffic to authority switches [132] that are responsible for monitoring the flows between the source and destination. LiteFlow distributes monitoring workload among switches and manages the switch resources. LiteFlow reduces flow rules that are installed in the switch [245].

The authors [246] propose two traffic matrix estimation approaches—Maximum Load Rule First (MLRF) and Large Flow First (LFF). MLRF generates flow rules that maximizes the traffic load in switch, using rule prioritization to move traffic between flow rules. LFF measures large flows in the network to get an overall TM estimation. The authors [246] use MLRF rule statistics to identify the large flows. Feasible and accurate TM estimation are achieved with MRLF and LFF [246].

The solution [247] uses a separate controller to gather statistics and to identify large network flows. The solution [247] performs tradeoffs [239, 240] between accuracy and statistic gathering overhead of switch. There are three measurement primitives—memory counter, hash data structure, and measurement program on the switch that allows statistics gathering [239].

Routing

The authors [248] use source-based routing to reduce network state required by the controller to maintain and distribute, increasing scalability of the controller. The solution [248] pushes the network state of the controller to edge switches. The controller sends the edge switch a sequence of interfaces or path that the flow traverses. The edge switch forwards the packet with path appended in header through the network. Each switch inspects the packet header and forwards the packet out of interface.

The amount of controller state reduction is proportional to the links in the network path [248].

The authors [249] measure packet forwarding delay and convergence time after link or node failures between legacy routing protocols and SDN routing [250]. The response time of legacy network is more than SDN network for large networks and SDN network response time is more than legacy network for small networks [249]. The routing convergence time of legacy network is influenced by link delay [249]. SDN network avoids transmitting network information between switches; therefore, link delay does not influence SDN convergence time. The switch only maintains relevant state in SDN which avoids the need to store the complete network topology in the switch, improving the forwarding speed of switch [249].

2.4.5 Level of Programmability

This section discusses research APIs which contribute to the characteristic of level of programmability. The API gives the network operator ability to program and control the behavior of network.

Simplified Wrapper and Interface Generator (SWIG) [251] is a software development tool that enables programs written in C/C++ to connect with a variety of high-level programming languages such as JavaScript, Python, and Ruby. SWIG creates high-level interpreted or compiled programming environments and user interfaces. SWIG is used for testing and prototyping C/C++ software.

OpenFlow [21, 252] is a communications protocol that gives the control plane access to data plane. The control plane sends modification messages to the switch over a secure channel. OpenFlow allows researchers to experiment with network protocols without the network vendor exposing switch details, allowing the behavior of network to be changed programmatically by the network applications and services. OpenFlow increases complexity of the packet processing of switch [253].

OpenStack [254] is a free and modular open source stack for developing cloud computing fabrics, cloud controllers, and cloud applications. OpenStack allows network operators to build rich network topologies, such as layer-2(L2)-tunneling-layer-3(L3). OpenStack enables automation and orchestration of cloud resources and cloud applications. OpenStack is a collaboration of more than one-hundred and thirty companies from the server and application domains.

Big Switch Networks developed Floodlight Northbound API [255], a RESTful API that lies between the northbound and southbound APIs of the controller. Floodlight Northbound API enables maximum network utility and allows network operator to interact with the network. Floodlight Northbound API allows network applications and services to communicate with the control plane [255].

Virtualization APIs are available within the hypervisor on client machine. Virtualization APIs are used by network operators to control and manage virtual machines on client machine. Virtualization APIs are upper layers that communicate with the controller and switch. Virtualization APIs allow automation and optimization of network servers, storages, and configurations.

2.5 SDN Research Challenges

2.5.1 Scalability

In medium sized networks, a single controller processes control plane events—packets effectively [152]. In large networks, such as carrier networks, a single controller lacks the ability to process substantial control plane events in a timely manner. Achieving scalability with a single controller in carrier grade networks is hard because of the requirement to achieve link recovery in 50ms or less [95]. Distributing the control plane events over multiple controllers ensures timely responses from the controller to network infrastructure—switch and middlebox.

Distributed controllers [82, 148, 151, 256–258] have been developed to increase the scalability of control plane. Controller farms dynamically provision [258] controllers

on demand [259] from a cluster of controllers and load balance control plane events across multiple controllers, reducing the burden on a single controller [260]. The SDN architecture maximizes the network events processed in data plane [134] and minimizes network traffic between distributed controllers [261], improving the scalability of the controller.

The high cost and energy requirements make flow table expensive and limit flow rules that can be installed. The rule installation should be performed efficiently [262]. The two main approaches for managing flow table space are compression and caching [186, 263]. Although flow rule optimization [263] and compression techniques are researched [133], the switch flow rule space remains a scalability issue of the SDN architecture. Increasing flow table size by using memory and CPU of the switch [264] can be performed, but longer packet delays are introduced when using the memory and CPU of switch.

2.5.2 Availability

The network operator deploys multiple distributed controllers [82,148,151,258] to ensure high availability [265,266] of network services. The distributed controllers create the global network state using a proof labeling scheme to increase the availability of network in the control plane [261].

The solution [113] increases the availability of data plane by using a Chord assignment policy to install backup flow rules. The ability to recover during link failures [111, 112, 122, 267–272] and reliably less than 50ms directly impacts availability [95].

For network applications, high availability [63] may mean QoS such as bandwidth, delay, jitter, and reliability is guaranteed. There may be unacceptable delays between the control plane and data plane [273]. Achieving high availability requires redundant hardware in the data plane [274] and control plane and intelligent software such as load balancers [182, 187] and failover functionality [113, 267].

2.5.3 Security

Protecting the controller allows network services to function without interruptions. The switch communicates with the controller using southbound communication over TLS. TLS has its own inherited security vulnerabilities such as man in the middle attack. Distributed controllers—multi-domain SDN [60, 124, 166, 273, 275–279] communicate with one another using east/west-bound communication. Recent research propose securing distributed SDN communication with multi-domain capable Identity-Based Cryptography (IBC) protocol [280].

The orchestration layer [51] helps prevent attacks [281] from misbehaving applications. Security has a direct impact on the availability and reliability of network services [282]. Distributed state and control enhance security by ensuring no single controller has complete network state and control.

In SDN, a key vulnerability is installation of conflicting flow rules of switch. To prevent routing loops [214, 283], security vulnerabilities, network outages, and ensure consistent processing of packets, the switch flow table should be consistent with network policies [195], avoiding misconfiguration of switch [284]. Flow rule conflicts among dynamic applications should be resolved to reduce security risk [192].

Security [193] is increased by using logging, recording and playing back network events [74], and continuously monitoring [48, 226] the network.

2.5.4 Standardization

SDN APIs are still essentially proprietary although there are standardization efforts for the OpenFlow [21] switch. The OpenFlow Switch Consortium was established in 2008 to maintain the OpenFlow switch specification [285]. The switch functionalities and performances varies among vendors [160]. There are significant differences between OpenFlow specifications [70] which affects application performance.

The Open Networking Foundation (ONF) [16] was established in 2011 in an effort to increase awareness about OpenFlow and to promote commercialization of SDN.

OpenDaylight [286] was later created by the networking industry to help standardize the SDN platform. The interfaces of controller are still in the early stages and independent from one another [287]. A standardize controller platform helps with adoption of SDN. At the time of writing, there is no standardized controller platform.

2.6 Networking Industry

Network vendors are pushing SDN from a concept to implementation of a network solution which is scalable, automatable, and optimizable in data centers. Data centers are composed of physical and virtual networks that are difficult to manage because of low level network details such as ports and links. Managing low level network details is becoming less efficient with traditional approaches. The network operators will no longer accept unwillingness of network vendors to change [30]; therefore, network vendors are required to provide an efficient solution to manage and to optimize the network. There is a market for providing a network solution which eases the life of network operators—SDN [288].

IBM [31] developed a 10Gb OpenFlow switch [289] and an OpenFlow controller which provides centralized control over flows and unlimited virtual machine mobility [290]. IBM [31] enables deployment of efficient centralized networks, increases network controllability, and allows the network to be dynamic and flexible to meet business needs [291].

Extreme Networks OneFabric [32] provides real-time configuration of virtualized network resources and bridges the gap between virtual machines and network applications. OneFabric implements locationing and provisioning services in converged networks to ease burden of network operator.

Oracle SDN [292] enhances application performance and management by dynamically connecting servers and VMs to networks, storage devices, and other VMs. The Virtual Network Services feature of Oracle SDN provides the ability to rapidly de-

ploy secure on-demand network services such as firewall, router, load balancer, Virtual Private Network (VPN), and NAT in a single virtual appliance [292].

Big Switch [26], the leading platform-independent SDN vendor, developed Open SDN architecture which includes the Big Network Controller and Big Virtual Switch. The Big Network Controller includes network applications that allow the network operator to manage, automate, and optimize data center networks.

HP FlexNetwork Architecture provides application characterization, network abstraction, and automated orchestration. HP developed the HP Virtual Application Networks SDN Controller [24] which provides a network abstraction and automates orchestration of cloud services. The HP controller enables customers to migrate to the cloud and allows cloud providers to leverage the benefits of SDN. HP Intelligent Management and Virtual Application Networks allow businesses to create scalable, agile, and secure networks [24].

Cisco Open Network Environment (ONE) [293] provides an integrated solution that makes the network open, programmable, and application-ware. ONE [293] optimizes network resources, reduces network operational cost, reduces network misconfiguration, and accelerates network service delivery.

Pica8 [294] developed an open SDN reference architecture that addresses cloud service providers need to reduce capital expense and control operational expense. Pica8 developed PicOS, a white box switch operating system that is hardware agnostic [294]. PicOS supports OpenFlow and OpenStack and fits [295] into existing networks.

NEC developed Programmable Networking [25], a SDN solution for data centers. Programmable Networking [25] creates a cloud-ready network which is fast, scalable, and open. NEC ProgrammableFlow Networking Suite was the first commercially available Software-Defined Network (SDN) solution to leverage the OpenFlow protocol, enabling full network virtualization [146,170–172] and allowing network operator to deploy, control, monitor, and manage secure multi-tenant networks [25].

Metaswitch has taking an evolutionary approach to SDN [296]. Metaswitch focuses on using SDN to increase operations and capabilities of Multiprotocol Label Switching (MPLS) and Generalized Multi-Protocol Label Switching (GMPLS). Metaswitch uses Path Computation Elements (PCEs) to provide the network operator with the benefits of SDN while leaving existing network equipment to run mature protocols and algorithms. Metaswitch removes path computation function from the network and places the function in centralized PCE server; other existing functions of the network devices remains on switch.

Ericsson [29] Service Provider SDN is making SDN a reality for service providers [28]. Ericsson [29] is using cloud, Network Function Virtualization (NFV), and SDN to transform the network, making network programmable, automatable, flexible, and application-responsive [297]. Ericsson introduced the concept of virtual Customer Premise Equipment (CPE) [298] that allows network services to be moved from home user router to the cloud. The ability to move network services from router to the cloud enables dynamic service chaining with SDN [299]. Ericsson Dynamic Service Chaining solution uses SDN technology to chain network functions where traffic from subscriber traverses a particular set of service functions [299].

Rapid growth of the SDN market depends upon timely and broad support of a core set of APIs across controllers of multiple vendors [300]. Evidence of the networking industry strong commitment to adoption of SDN is available OpenFlow enabled switches [287]. The ability to integrate [109, 178] and control [301] legacy switches as SDN is deployed aids in adoption of SDN.

Adoption of SDN is slowed by absence of a standard, decreasing interoperability [302] among network vendors. Each vendor has its own API and SDN functionality that limits the ability to engineer and manage traffic across equipment from multiple vendors [303]. As SDN is still nascent, standard protocols of the networking industry are still emerging, but moving forward it is important these standards get created [304].

Table 2.1: Classification and contribution of SDN research A-N

Project	**Application Domain**	**Layer of Control**	**Level of Programmability**	**Network Technology**	**References**
AutoI	Network virtualization	Data plane and control plane		Internet	[339]
Ca-SDN	Routing	Application plane, control plane, and data plane	OpenFlow	Cloud	[340]
DevoFlow	Traffic classification	Control plane	OpenFlow	Data center	[134]
DIFANE	Traffic directing	Data plane and control plane	OpenFlow	Enterprise	[132]
DISCO	Traffic engineering	Control plane	OpenFlow	WAN	[60]
Elastic Tree	QoS energy	Data plane	OpenFlow	Data center	[341]
Fleet	Security	Control plane	OpenFlow	Enterprise	[175]
Floodlight		Control plane	OpenStack and OpenFlow	Enterprise	[102]
FLOWGUARD	Network policies and security	Control plane	OpenFlow	Enterprise	[194]
FlowSense	Link utilization	Control plane	OpenFlow	Internet	[243]
FortNOX	Network policies and security	Control plane	SWIG	Enterprise	[192]
FRESCO	Security IDS and IPS	Application plane	OpenFlow	Enterprise	[191]
Hedera	Flow scheduling	Control plane	OpenFlow	Data center	[180]
HyperFlex	Network virtualization	Control plane	OpenFlow	Carrier	[168]
HyperFlow	Network synchronization	Control plane	OpenFlow	Data center	[151]
Kandoo		Control plane	OpenFlow	Data center	[82]
LIME	VM migration	Application plane	OpenFlow	Cloud	[179]
NetCore	Functional reactive programming	Control plane	OpenFlow	Enterprise and data center	[209]

Table 2.2: Classification and contribution of SDN research O-Z

Project	Application Domain	Layer of Control	Level of Programmability	Network Technology	References
OFLOPS	Application and switch performance	Control plane	OpenFlow	Carrier	[159]
OFRewind	Switch configuration	Control plane	OpenFlow	Data center	[74]
Open Programmable Extensible Networks (OPEN)	Traffic engineering	Data plane	OpenFlow and Virtualization API	Internet	[183]
Open Router Virtualization Framework	Network virtualization	Data Plane	OpenFlow	Internet	[342]
Open Transport Switch (OTS)		Data Plane	OpenFlow	Internet	[119]
OpenVirteX	Network virtualization	Control plane	OpenFlow	Enterprise	[167]
OpenvSwitch		Data plane	OpenFlow		[118]
OpenADN	Network Policies	Data plane	OpenFlow	Internet	[343]
OpenFlow		Data plane			[252]
OpenSAFE	Traffic directing	Control Plane	OpenFlow	Enterprise	[244]
OpenTM	Traffic matrix	Application plane	OpenFlow	Enterprise	[241]
Procera	Functional reactive programming	Application plane and control plane		Enterprise and home	[205]
QoS-aware Network Operating System (QNOX)		Control plane	OpenFlow	Carrier	[344]
QuagFlow	Routing	Application plane and control plane	OpenFlow		[345]
RouteFlow	Routing	Application plane and control Plane	OpenFlow	Internet	[346]
SOFT	Switch configuration	Data plane and control plane	OpenFlow		[161]
SoftRAN	Load balancing and mobility	Control plane		Cellular	[91]
SplitArchitecture (SPARC)		Application plane and data plane		Carrier	[94]
VeriFlow	Network policies	Control plane		Internet	[214]

2.7 Conclusions

This chapter discussed the state-of-the-art in SDN. It reviewed past programmable network research efforts, and discussed how they impacted SDN. This chapter reviewed and summarized SDN research. This chapter identified future SDN research challenges. It discussed the state of SDN in networking industry. It discussed network vendors and how the vendors are pushing SDN because of SDN's ability to reduce expenses. This chapter presented a set of characteristics and classification scheme for better understanding about relationship between existing SDN bodies of work. The classification scheme includes.

- **The network technology**—implicitly determines the level of programmability.
- **Layer of control**—the network plane that the network operator controls behavior of network.
- **The application domain**—indicates functionality of application such as routing and load balancing.
- **The level of programmability**—indicates the network plane where network service is exposed and API that accesses the network service.

SDN allows rapid innovation of network applications, reduces network capitol expenses, and controls network operation expenses. This chapter identified key innovations that are pushing networking paradigm shift. The innovations are leading to a programmable, automatable, and flexible network known as SDN. The key innovations pushing network paradigm shift are.

- ***virtualization*** of network and network hardware such as servers and storage,
- ***separation*** of the data plane and control plane—network hardware and network software,

- ***availability*** of open programmable interfaces such as OpenFlow and OpenStack,
- ***rapid development*** and ***rapid deployment*** of network services such as firewall, NAT, and DPI, and
- ***automation*** and ***full control*** of network services such as billing systems and load balancing.

SDN is a network architecture that supports virtualization of computer network and network hardware, separation of the control plane and data plane using an open programmable interface, and rapid development and rapid deployment of fully automated network services.

3 VIDEO OVER SOFTWARE-DEFINED NETWORKING (VSDN)

3.1 Abstract

Supporting end-to-end quality of service (QoS) for video applications requires the network to select optimum path among multiple paths to improve the performance of video application. Multiple paths between source and destination may be available, but because of the network high coupling design identifying alternative paths is *difficult*. Network architecture such as Integrated services (IntServ) installs path from source to destination that may not be optimum—best case path for the video application. Furthermore, it is an arduous task for video application developers to request service from IntServ.

This chapter provides three contributions to the literature on providing end-to-end QoS for real-time interactive video applications. This chapter presents Video over Software-Defined Networking (VSDN), a network architecture that selects optimum path using the network-wide view. This chapter describes how the video application developer uses protocol for requesting service from the network. This chapter presents the results of implementing a prototype of VSDN, evaluating behavior of VSDN. Requesting service from VSDN requires three parameters from the video application developer. The message complexity of VSDN is linear.

3.2 Introduction

Integrated Services (IntServ) framework, a flow based Quality of Service (QoS) network architecture, allows network elements such as sender, receiver, and routers to reserve network resources which guarantees end-to-end service. IntServ framework uses two protocols—flow specification describes traffic patterns and reservation pro-

tocol transmits reservations among network elements and allows applications such as video and voice to make reservations. The QoS sensitive tasks such as video compression and audio compression of the applications require guaranteed bandwidth and bounded delay and jitter.

Supporting end-to-end QoS for real-time interactive video applications requires the network to select optimum path among multiple paths to improve the performance of video application. The network control protocols may not install optimum path for the video application and is incapable of exploring alternative paths since the control protocol relies on the routing protocol to select path. For example, if there exists two network paths—path1 and path2; path1 that has congestion is 3 hops and path2 that has no congestion is 5 hops. Path1 that is 3 hops is selected to install reservation if routing protocol uses shortest path. Selecting path1 for real-time interactive video applications negatively impacts the performance of video application because path1 is congested.

The network path selection should be adaptive to changing network conditions such as link failures and node failures and should be context aware about state of paths including bandwidth, jitter, and delay. For example, network path selection can be automated based on characteristics of certain applications [305]. Although MPLS, data-carrying mechanisms, selects multiple paths, MPLS is not dynamic and the network operator manually configures paths at each router within the network [197]. The network control protocols are capable of adapting to network failures, but data packets are lost or receive best-effort service between failure and next PATH refresh messages which is 30 seconds before path is updated and resources are allocated to network flow; therefore, this chapter presents the experience and results of identifying a network architecture that ensures end-to-end QoS for real-time interactive video applications.

The main contributions to research are. This chapter presents Video over Software Defined Networks (VSDN), a network architecture and protocol for supporting real-time interactive video applications. This chapter illustrates how the sender and

receiver use the application program interface (API) to request service from the network. This chapter presents the results of implementing VSDN in a network simulator. This chapter evaluates VSDN runtime performance using message complexity.

Chapter organization. The remainder of this chapter is organized as follows. Section 3.3 motivates the need for Video over Software-Defined Networking (VSDN). Section 3.4 discusses the design and implementation of VSDN. Section 3.5 presents and interprets the simulation results. Section 3.6 compares VSDN to related works. Section 3.7 provides concluding remarks and lessons learned.

3.3 Motivation: Integrated Services (IntServ)

Integrated Services (IntServ) network architecture uses reservation protocols to signal end-to-end QoS over IP networks. The network resources are reserved on a hop-by-hop basis using two messages. The sender uses the PATH message to install reverse routing path on each router along the path and conveys to the receiver characteristics of expected network traffic. The receiver uses the RESV message to request QoS of packets from each router along the path. Reservation protocols use PATH and RESV messages to install soft states in the network devices along the path selected by routing protocol. The soft states contain descriptions of the expected network traffic characteristics such as traffic rate, queue size, and peak traffic rate. The reservation protocol works with different routing protocols to provide QoS for real-time applications.

The resource reservations are made using the path that the routing protocols such as RIP, IS-IS, or OSPF select. Reservation protocols have advantages such as soft-state adaptive nature, the ability of receiver to initiate the reservation, and the possibility to merge reservation requests. The reservation protocol which is used by IntServ has its disadvantages. For example, in Figure 3.1, the best path—bandwidth, jitter, and delay to send the video traffic is R1-R2-R4-R5. If routers in autonomous system one (AS1) is running OSPF, the shortest path stored in each router link state

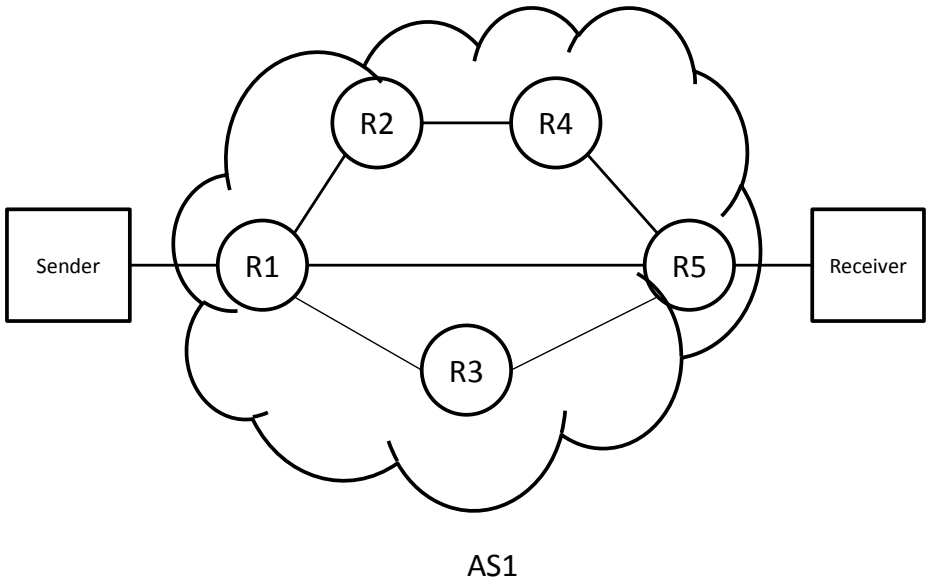

Figure 3.1.: A network with sender and receiver.

database is R1-R5; therefore, packets to and from the sender and receiver are sent along path R1-R5 which is two hops.

The reservation protocols install PATH and RESV states in R1 and R5 to support QoS for real-time interactive video applications; although the best path for the video traffic is R1-R2-R4-R5. InServ is unable to select the best path for the video traffic. Furthermore, if a link failure occurs between R1 and R5, a new path is found. The network makes the following adjustments. A link failure is detect and each link state router database is updated to reflect link R1-R5 failure. The next shortest path R1-R3-R5 is selected after failure, but the best path for the video traffic is R1-R2-R4-R5. The network has failed to find optimum or best path for the real-time interactive video application.

Because relying on the network routing and reservation protocols to provide end-to-end QoS for real-time interactive video applications may not deliver the best performance, there are two issues that this chapter addresses.

Issue 1: Identify network architecture that supports optimum path selection. As Figure 3.1 illustrates, selecting optimum or feasible path requires network-wide view or global network state. Hop-by-hop decision making may not deliver the best performance. A network architecture needs to be developed that dynamically selects optimum path and provides feasible backup paths in the case of failure—node or link.

Issue 2: Develop protocol that allows real-time interactive video applications to request end-to-end QoS from the network. A protocol needs to be developed to allow the video application developers to request network services from the network. The protocol needs to request and maintain state that is necessary to ensure optimum path selection.

The rest of this chapter discusses how Issue 1 and Issue 2 are addressed. This chapter integrates usability when designing and implementing VSDN.

3.4 Design and Implementation

One key design requirement for addressing—Issue 1 : Identify architecture needed to support optimum video path selection is the need for the network-wide view [306] to make optimum path selection; this requirement leads to the use of software-defined networking (SDN) [16] and OpenFlow [21] to address Issue 1.

Figure 3.1 has been redesigned to use SDN and the OpenFlow protocol as illustrated in Figure 3.2. A SDN controller that has the network-wide view has been added.

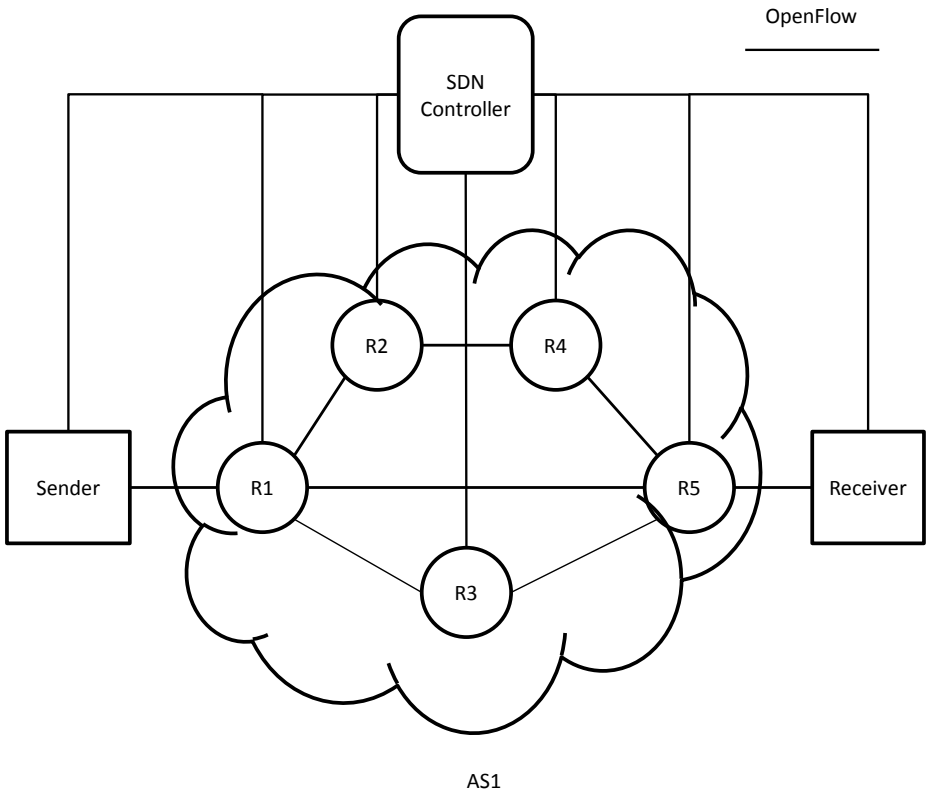

Figure 3.2.: SDN network with sender and receiver.

3.4.1 Software-Defined Networking (SDN)

Figure 3.2 illustrates the SDN architecture. SDN separates the control plane—routing decision from the data plane—forwarding decision. The control plane is logically centralized in the controller and runs on commodity hardware. The switches become dumb devices, performing packet forwarding with instructions from the controller. The OpenFlow protocol enables communication between the control plane—controller and the data plane—switch.

3.4.2 VSDN Design Overview

VSDN is a network architecture that allows real-time interactive video applications to request guaranteed service from the network. The design of VSDN integrates usability.

Figure 3.3 illustrates the VSDN architecture. The architecture has four system elements—sender, switch, controller, and receiver. The sender and receiver rely on the network that is composed of R1 and R2 to provide end-to-end QoS.

The Policy Control (PC) is separated from physical device—router or switch and is relocated within Video QoS Controller (VQC), providing policy consistency among the network devices. The PC accepts commands from the network administrator about how to process the network traffic. The network administrator uses the PC to enforce network constraints [284]. The policy translator maps policies stored in policy database to network configurations. The Resource Monitor (RM) monitors the network resources. The RM periodically collects statistics from the physical devices—switches and middleboxes. The RM stores the network state in the resource database.

The VQC uses the Admission Control (AC) to reserve network resources when QoS request is received. When a request is received on a router interface, admission control is performed on the interface. If network resources are available on the receiving network interface, the video QoS process finds an optimum or feasible path that

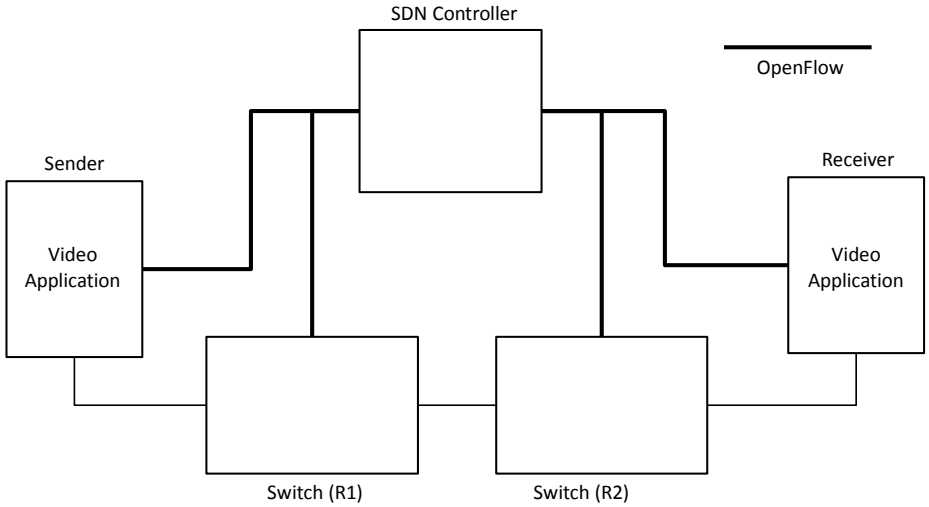

Figure 3.3.: VSDN architecture, showing the relationships between the architectural elements. There are four elements including the sender, switch, controller, and receiver. The sender and receiver rely on the switches, R1 and R2, to provide end-to-end QoS. The controller communicates with the switches, sender, and receiver over secure channels using OpenFlow. The sender and receiver request QoS from the network. The network devices R1 and R2 are edge switches which use packet shapers to shape traffic of the sender and receiver. A number of intermediate switches may exist between R1 and R2 which network traffic passes through, but only the edge switches shape network traffic. For simplicity, only R1 and R2 are shown.

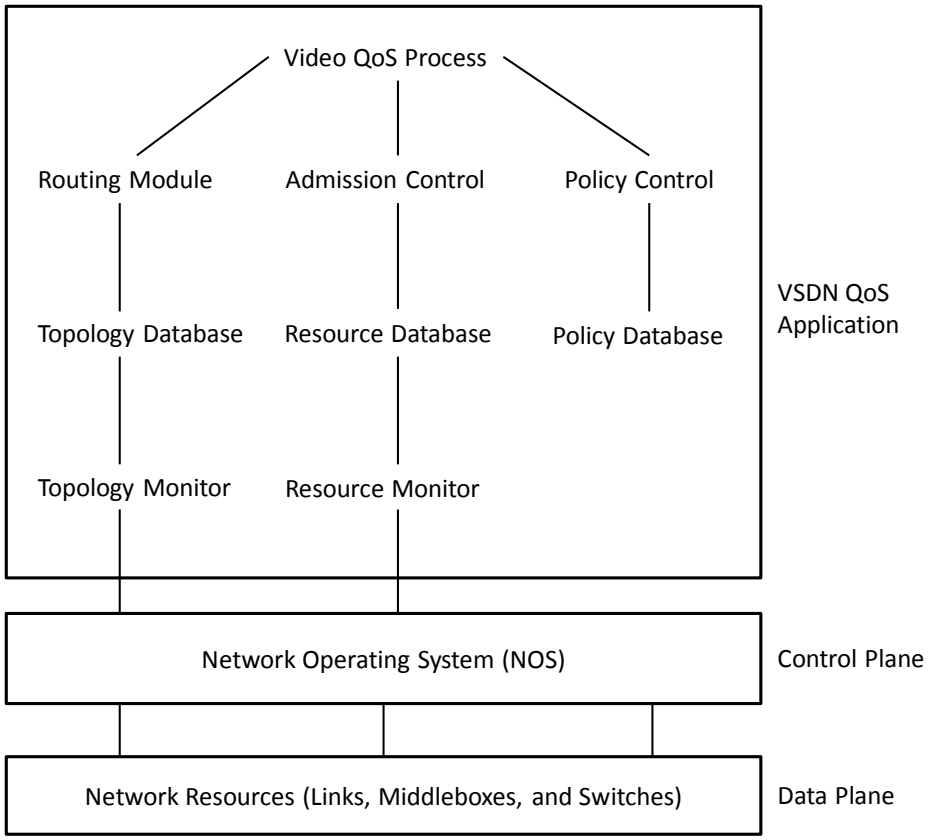

Figure 3.4.: VSDN controller, illustrating the architectural elements. The controller processes the QoS request. The controller manages the network resources such as bandwidth. The admission controller manages the network resources. The routing module finds feasible end-to-end paths.

can service the request. If network resources are unavailable on the interface, the VQC returns an error to the requester. The AC manages a pool of resources such as interfaces, bandwidth, memory, and CPU. The network resources are subtracted from the pool when a request is serviced and the resources are added to the pool when request finishes.

The Routing Module (RM) calculates feasible paths from ingress router to egress router. Constraint based routing algorithms and implementations have been studied in detail [307]. The RM returns a list of subgraphs or paths that meet the QoS constraints such as bandwidth, jitter, and delay. The Topology Monitor (TM) updates the network configuration when there is a network change such as node or link failure, deletion, or addition. The RM uses the network topology that is stored in a database to find feasible paths.

The main element of the architecture is the Video QoS Process (VQP), in Figure 3.4. The VQP processes the VSDN messages. The VQC processes sender, receiver, and error messages. The VQP maintains the session states and ensures network clients follow the VSDN protocol. If a network client fails to follow the VSDN protocol, the VQP generates an error stating the reason request failed. A valid request from sender generates a session in the session database. The session database contains flow information such as session identifier and destination and source addresses and ports. After a request is received from the sender, the request is forwarded to the receiver. If receiver accepts request or reservation, the receiver sends a request message to the VQC. Upon receiving a valid receiver request message, the VQC performs policy control and admission control to determine if the reservation can be made.

If the reservation is allowed, the VQP requests feasible paths from the RM. The RM returns a subgraph that the controller uses to configure network devices. After the network devices are configured, the receiver forwards confirmation message from VQC to sender, reserving the path. The sender and receiver communicate over the reserved path. The sender and receiver can issue the remove message to remove session from the network and release network resources. The VQP manages sessions and ensures

the sessions are updated or timed-out and removed. Figure 3.4 illustrates VQC running on network operating system [55] that provides a centralized programmatic interface to the network.

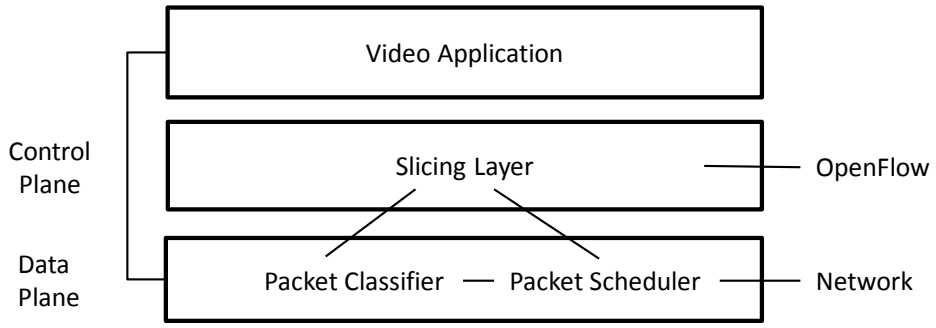

Figure 3.5.: VSDN client, showing the relationship of the elements. The network client has a slicing layer which is used for sharing home network. The controller uses slicing layer to configure network client QoS. The packet classifier identifies packets which belong to a specific flow. The packet scheduler ensures packets are in profile before entering the network.

Figure 3.5 illustrates the architecture of network clients—the sender and receiver. The VSDN enabled sender and receiver host the video applications. The slicing layer [99] allows multiple service providers to share common infrastructure and supports more than one policy and business model. The slicing layer allows the VSDN service providers to control their share of the client network. The video application communicates with the network using QoS API. The packets belonging to a flow are identified by the packet classifier. The packets are identified using the address of sender, address of destination, port of sender, port of destination, and protocol ID.

The packet scheduler ensures the video packets are in specification such as rate, bandwidth, and queue size. The packet scheduler ensures packets are in specification before the packets are transmitted to the network. The network administrator has global policy control over the network elements—client, switch, and router.

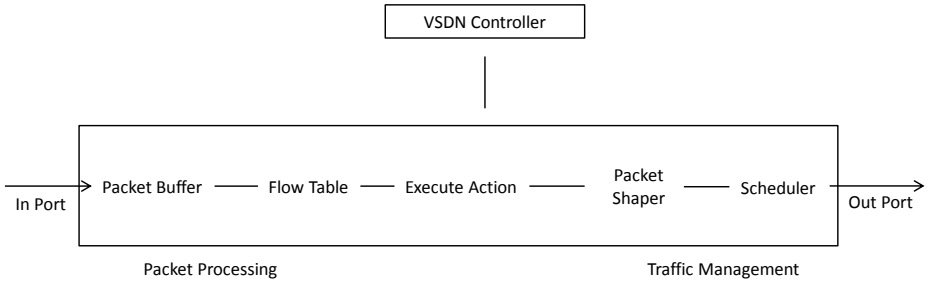

Figure 3.6.: VSDN switch, showing the relationship of the elements. A packet enters the switch through the In Port and continues through pipeline. The packet is dropped, forwarded to controller, or forwarded out of Out Port at Execute Action. The port has a packet shaper—a queue that ensures QoS of each flow. Only edge switch has packet shaper installed because the edge switch shapes the network traffic.

Figure 3.6 illustrates the architecture of VSDN switch—R1 and R2. The VSDN switch is OpenFlow enabled. The VSDN controller upon receiving and validating QoS service request from network client issues set-queue action to each switch along the path. The controller issues flow-add action to each switch along the path after issuing the set-queue request. If either set-queue or flow-add request fails, the switch returns an error message to the controller. The set-queue request causes the switch to create a per-flow Weighted Fair Queue (WFQ) and traffic shaper using Traffic SPECification (TSpec) [308].

In Figure 3.6, a packet enters the switch and is queued. The packet is forwarded through flow table pipeline where the packet is sent to the controller or dropped or continues through the flow table pipeline until there is a matching flow table entry. The packet is eventually forwarded to the port where QoS has been configured. The configuration of queue dictates basic QoS forwarding behavior. Full end-to-end QoS support over SDN is unsupported in OpenFlow [21].

3.4.3 VSDN Protocol

This section discusses the action of each element of the VSDN architecture and how the sender and receiver interact with the network.

Sender

The sender, in Figure 3.2, makes a call to QoS API using requestQoS(video service, destination address, destination port). The video applications remove session from the network using the Session Id returned from requestQoS. The video application developer can request four video services—Common Interchange Format (CIF), Enhanced Definition (ED), and High Definition (HD).

The router R1 forwards message to the VQC after receiving the message from sender in Figure 3.2. The VQC determines if request of sender exists. If session does exists, the VQC responds with invalid request, if the message is not from a timeout

request. If the session does not exist and policy allows sender to issue request, the VQC creates the session and responds to R1 that forwards the packet to receiver through R2. The sender waits on the confirmation from receiver. After receiving the confirmation from receiver, the sender and receiver begins the video session.

Receiver

The receiver receives request from the sender on an application callback processRequest(SessionID) and call acceptQoSRequest(SessionID). The video application stores the SessionID that is used for accepting a request and removing a session from the network. The receiver makes reservation with the network using traffic pattern described by the sender. R2 receives request from the receiver and forwards the request to the VQC that determines if the session exists. If the session does not exist, the VQC generates an error that is sent to receiver. If session does exist, the VQC uses the policy control, admission control, and router module to make the resource reservation, if the receiver is authorized.

The RM returns feasible paths for the request. The VQC selects the route and issues configuration request to the network operating system (NOS) which manages the network devices. The receiver receives the confirmation from VQC about the status of installed path. After receiving the confirmation from VQC, the receiver sends the confirmation to sender. The VQC returns an error message to the receiver if the VQC is unable to find a feasible path. The path is reserved and configured when the sender receives the confirmation message.

Removing Reservation

The network client explicitly removes reservation from sender, receiver, and network using removeQoSRequest(SessionID) or implicitly through flow timeout from the switch or network client. The edge switch forwards the remove message from sender or receiver to VQC where validity of message is determined. The VQC after

validating message removes the session from session database and flow entries from the switches and network clients. The edge switch forwards the remove message from the VQC to destination. The flow timeout implicitly removes the flow from network client if the remove message is not received from the network.

Controller

The VQC or controller receives request from the receiver. After receiving request from receiver, the controller performs admission control and policy control. Policy control determines if the receiver can make reservation and admission control determines if network resources are available to service request. The controller finds the path that can service request. After finding the path, the controller installs the flows and queues in the switches and network clients. The controller forwards a confirmation message to the receiver. The receiver responds to the sender with a confirmation message.

Switch

The VSDN switch is a network device that is programmed by the controller. The switch forwards the request message to the controller. The controller determines if a reservation can be made by the receiver. The switch installs the flow in flow table of switch and installs the queue on port or return an error to controller if the switch is unable to complete flow and queue installation.

Network Client

The VSDN network client is a network device that is programmed by the controller. The controller sends the client flow modification request which client uses to configure its flow table. The client performs instructions—add flow or delete flow and install queue on port when programmed by the controller. The client returns an

error to the controller if the client is unable to complete flow modification or queue installation.

3.4.4 OpenFlow Changes

An OpenFlow switch provides limited QoS support through a simple queuing mechanism [21]; therefore, the VSDN switch requires changes to OpenFlow queue structures.

The queue properties are modified to support guaranteed service (GS). A new property is added to support GS queuing. The GS queue property contains fields such as traffic rate, bucket size, peak traffic rate, minimum packet size, maximum packet size, maximum link capacity, and rate for token bucket traffic shaping. In Figure 3.6, the switch creates a token bucket shaping queue for each requested flow. The queuing process regulates traffic for each flow using the traffic specification provided by the controller.

3.4.5 Network Client API

The network client API allows the client to request service from the network. The interface of VSDN requires three input parameters from the developer. The details of API are not discussed in this chapter.

The service is requested using requestQoS(v, d, p). The sender can request three types of video services—Common Interchange Format (CIF), Enhanced Definition (ED), or High Definition (HD).

The receiver accepts the request using acceptQoSRequest(s). The sender and receiver can remove a session using removeQoSRequest(s). The processRequest(s, d) notifies application about network related events.

3.4.6 QoS Mapping

There are two service specifications for real-time tolerant applications such as video streaming and real-time intolerant applications such as interactive video. The two service specifications are Controlled Load (CL) [309] and Guaranteed Service (GS) [308]. The VSDN architecture supports GS. The VSDN protocol uses the attributes in Table 3.2 to configure the token bucket processes [310] of the network devices.

The application specifies the video type such as CIF, ED, or HD and avoids specific flow specification attributes. The controller understands the video types and converts from video type to TSpec and sends request to the network devices; therefore, a mapping scheme is needed to map between video type and TSPec.

Table 3.3 describes the mapping between video type and GS specification. The values were derived from available data from Google Hangout and Microsoft Skype communication platforms. The frame size in kilobyte (KB) determines packets and bucket size (b) needed to deliver a single frame.

For example, HD requires 60 KB for each frame which is 40 * 1.5 KB packets. At 30 frames per second (fps), video type HD has rate (r) of 1200 (fps * b) and bucket size (b) of 40 which holds a single frame. The slack term (s) is dependent on the service level agreement (SLA) between service provider and customer. The switch does not use slack term because the switch does not manage network resources. The slack terms enables the controller to make network resource adjustments.

3.5 Results

This section analyzes message complexity of VSDN.

Performance metrics - to assess the performance of VSDN, this chapter chooses following performance metrics.

- *Message complexity* - measures message count for the client request. The types of messages are illustrated in Table 3.4.

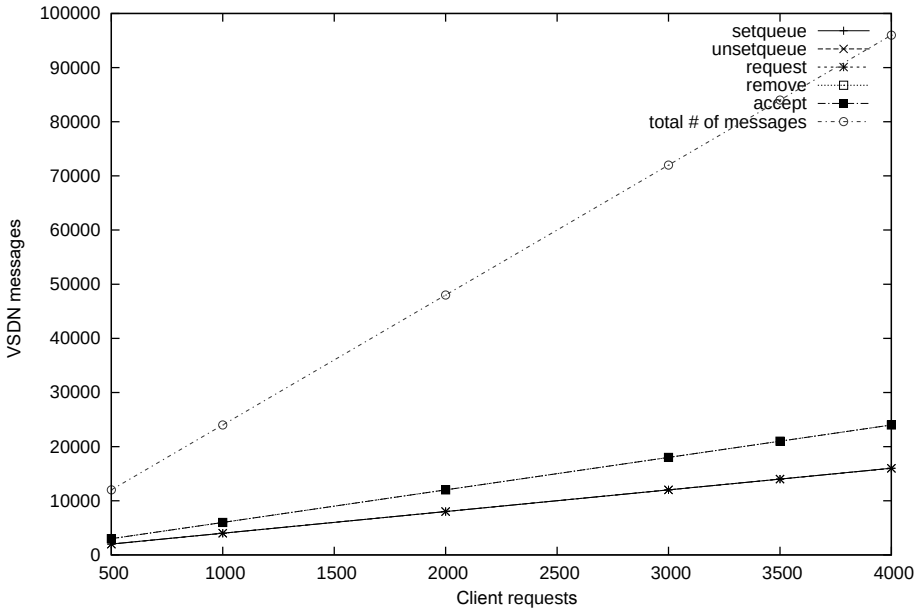

Figure 3.7.: Average VSDN message count when client requests increases. The six node network message complexity is linear.

Figure 3.7 illustrates messages in system with six-node network. At 500 client requests, the setqueue and unsetqueue messages are 2,000. The request messages are 2,000. The remove and accept messages are 3,000 because the remove and accept messages traverse the control plane for resource management. At 1,000 requests, the setqueue and unsetqueue messages are 4,000. The request messages are 4,000. The remove and accept messages are 6,000. Each message type increases linearly from 500 client requests to 4,000 client requests. The system message count for 500 client requests is 12,000. At 2,000 client requests, the system message count is 48,000. At 4,000 client requests, the system message count is 96,000. From the results, the message count is 24 times client requests. This chapter did not introduce network errors during test runs which affects message count including error messages which count is 0.

Figure 3.8 illustrates messages in system with thirteen-node network. At 500 client requests, the setqueue and unsetqueue messages are 2,500. The remove and

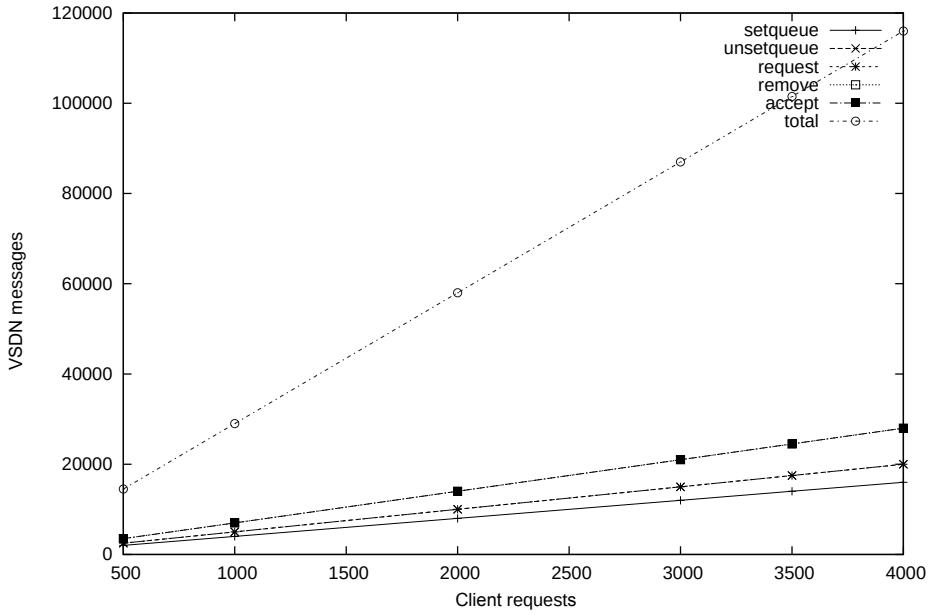

Figure 3.8.: Average VSDN message count when client requests increases. The thirteen node network message complexity is linear.

accept messages are 3,500. At 1,000 requests, the setqueue and unsetqueue are 5,000. The remove and accept messages are 7,000. At 2,000 requests, the setqueue and unsetqueue messages are 10,000. From results, each message type increases linearly from 500 client requests to 4,000 client requests. At 500 client requests, the message count is 14,500. At 1,000 client requests, the message count in system is 29,000. At 2,000 client requests, the message count in system is 58,000. At 3,000 requests, the message count in system is 87,000. At 4,000 client requests, the message count in system is 116,000. The message count is 29 times client requests.

Between Figure 3.7 and Figure 3.8, the message count in system is 24 and 29 times the client requests respectively. In Figure 3.8, the message count is 5 times more than message count in Figure 3.7 because of the longer paths between the source and destination with the thirteen-node network. Each additional node in path is configured with VSDN messages; therefore, the node count affects the message count of the system.

3.6 Related Works

IntServ [3] is flow based network architecture that uses reservation protocols to signal end-to-end QoS between the sender, network, and receiver. VSDN is similar to IntServ because end-to-end QoS path resources are reserved explicitly between sender and receiver. Unlike IntServ, VSDN does not require refresh messages to refresh soft-states installed in the network devices. Flooding of refresh messages is one disadvantages of IntServ that affects scalability [311]. VSDN selects optimal path using the requirement of the application. IntServ is unable to explore alternative paths; therefore, IntServ selects the same path as routing protocol to install QoS which may not deliver best performance.

Differentiated Services (DiffServ) [311] uses flow aggregation and hop-by-hop decision making process to address the scalability issues of IntServ. DiffServ applies network-wide set of traffic classes. The network operator classifies flows between the

sender and receiver in a predefined manner. A network device, when receiving a packet marked with DiffServ value, applies scheduling and shaping techniques using traffic class. The Type of Service header field in IP header allows traffic classification. Unlike VSDN, DiffServ is unable to guarantee QoS to the application because each network device or router is configured independently and network-wide policing is difficult because there is no network-wide view.

Multiprotocol Label Switching (MPLS) is a layer 2.5 label switching technique that inserts label for network prefix to allow routers to perform quick lookup of label instead of using longest prefix matching [312]. The label technique allows MPLS to perform faster packet classifications and forwarding. As with DiffServ, MPLS aggregates or classifies flows in the path. VSDN works on a per-flow basis and could aggregate flows from a single user to optimize network resources. Unlike VSDN, MPLS lacks real-time path configuration during adverse network conditions such as node failure, link failure, or network congestion [312]. VSDN makes real-time configuration changes as a result of adverse network conditions without prior knowledge about the traffic pattern of network.

3.7 Conclusions

This chapter presented Video over Software Defined Networking (VSDN), a network architecture that selects optimum path for real-time interactive video applications—improving application performance. The developer request service from the network using the network client API. This chapter developed a prototype to illustrate the functions of the network and analyzed the behavior of VSDN. The message complexity of VSDN is linear.

After conceptualizing, designing, and implementing VSDN in a simulator, the following lessons were learned:

- **Development of a network service.** Developing a network service requires focus on usability—users and tasks.
- **Using a single SDN controller.** Using a single network controller for large scale networks leads to performance issues.
- **Separation of control plane and data plane.** Separating the control plane and data plane allows flexible application design choices to be considered.

Table 3.1: The API for requesting, accepting, and responding to requests.

Name	Description
requestQoS(v, d, p)	generate a QoS request
acceptQoSRequest(s)	accept QoS request
removeQoSRequest(s)	remove QoS request
processRequest(s, d)	callback for application

Table 3.2: The guaranteed services (GS) flow properties.

Name	Description
Token Rate (r)	The rate that tokens fill bucket
Token Bucket Size (b)	The bytes that token bucket can hold before overflow occurs.
Peak Data Rate (p)	The maximum data rate in bytes per second
Minimum Policed Unit (m)	The minimum packet size in bytes
Maximum Packet Size (M)	The maximum packet size in bytes
Rate (R)	Maximum link capacity or peak rate
Slack Term (s)	Additive end-to-end delay that the sender can tolerate between nodes if a node modifies requested flow specifications

Table 3.3: The video type to service specification mapping, illustrating values for each service specification for video type, bandwidth in Mbps, bucket size in bytes, peak rate in Kbps, minimum policed unit in bytes, maximum packet size in bytes, rate in Kbps, and frames per second.

Video Type	Bandwidth	Bucket Size	Peak Rate	Policed Unit	Packet Size	Rate	Frames Per Second
CIF	1.0	7	168	74	1522	168	24
ED	1.5	20	500	74	1522	500	25
HD	3.0	40	1200	74	1522	1200	30
HDx	6.0	54	3240	74	1522	3240	60

Table 3.4: VSDN protocol messages.

Message type	Description
setqueue	used by the controller for adding flow to network device
unsetqueue	used by the controller for removing flow from network device
request	used by client for requesting service from the network
accept	used by client for accepting request from the sender
remove	used by client for removing session from the network

4 EXPLICIT ROUTING IN SOFTWARE-DEFINED NETWORKING (ERSDN): ADDRESSING CONTROLLER SCALABILITY

4.1 Abstract

Software-defined networking (SDN) promises a more flexible, automatable and programmable computer network. SDN separates the control plane and data plane. The control plane is placed in a logically centralized controller which hosts network applications such as traffic engineering, QoS, and firewall. The centralized controller creates a scalability problem when processing large control plane events—packets. Reducing network events processed in the control plane and only processing required network events is critical in addressing scalability concerns of the SDN architecture. This chapter addresses the controller scalability problem by introducing Explicit Routing in SDN (ERSDN), a routing scheme that reduces the control plane events—packets.

This chapter makes three contributions to the literature on reducing the burden on controller. This chapter presents ERSDN, a routing scheme that selects transit routers throughout network at edge routers. This chapter presents the design and implementation of ERSDN. This chapter evaluates the effect of ERSDN on the scalability of controller by measuring the network events processed in the control plane. ERSDN reduces the network events processed in the control plane by 430%.

4.2 Introduction

The Software-defined networking (SDN) architecture is a network architecture that separates the control plane and data plane [16]. The control plane of network devices is relocated to a logically centralized controller. The control and data planes communicate with one another over a secure channel using OpenFlow [21]. The

controller has a network-wide view which the controller uses to provision, configure, and manage network resources such as CPU, memory, flow tables, and bandwidth in the data plane—switches and middleboxes. The controller configures routes through the network by modifying the flow table of switches. The entries in the flow table determines the packet action such as drop, forward to controller, or forward to port. Packets that have no matching entries in the flow table are forwarded to the controller for processing. The controller can block packet flow or install a new flow in the switch—ingress to egress. Once the flows are installed, subsequent packets traverse the installed path. Once the packets have traverse network, the flow for particular packets is removed either explicitly by the controller or implicitly by timeout built in the network device.

In SDN, the first packet of each flow is forwarded to the controller by each transit switch. The controller processes packet and installs flow in switch before returning the packet to the switch to be forwarded to next transit switch in the path. The next downstream switch receives packet and forwards the packet to controller. The controller installs flow in switch flow table and returns packet to switch that forwards the packet to the next hop. This hop-by-hop process continues until the packet exits egress switch. The control plane events generated by a single packet is proportional to the hops the packet traverses. The control plane events become significant as packets enter and leave the network using reactive installation of flows [180]. The network operator can proactively [180] instead of reactively install packet flows to reduce control plane events.

Proactively installing flow entries may have great consequences. For example, the network operator has complete knowledge about network traffic which enters network. The network operator proactively installs packet flows in switch. Security is an issue because the controller loses flow visibility when the flows are proactively installed. The network applications such as billing and monitoring require the network event generated by first packet—processed in control plane to function. If a switch or port fails, the network and controller is not configured to make adjustment for the failure;

therefore, packets would be lost. The network user would experience disruption in network service. There is a trade-off between flow table space and visibility of the network activity when flows are installed proactively. Although proactively installing flows reduces control plane events, proactive installation of flows is *hard* because the network operator cannot possibly know all network traffic patterns.

The SDN architecture flow installation protocol does not address large network events processed in the control plane—controller. The flow installation occurs on a hop-by-hop basis. Each switch forwards first packet of flow to controller. The unnecessary forwarding of the first packet increases control plane events.

Therefore, this chapter presents the experiment and results of Explicit Routing in Software-Defined Network (ERSDN), a reactive flow installation protocol used in Video over Software-Defined Networking (VSDN), a network architecture that provides end-to-end QoS for real-time interactive video application [200]. The main contributions of this chapter are. This chapter presents Explicit Routing in Software-Defined Networking (ERSDN), a flow installation protocol that uses explicit routing to reduce control plane events. This chapter presents the results of implementing ERSDN in a network simulator [313]. This chapter presents an empirical study that evaluates ERSDN runtime performance.

Chapter organization. The remainder of this chapter is organized as follows. Section 4.3 motivates need for Explicit Routing in Software-Defined Networking (ERSDN). Section 4.4 discusses design and implementation of ERSDN. Section 4.5 presents results of simulating ERSDN in a network simulator and interprets results. Section 4.6 compares ERSDN to related works. Section 4.7 provides concluding remarks.

4.3 Software-Defined Networking (SDN) Overview and Video Over Software-Defined Networking (VSDN) Implementation

This section discusses SDN/Openflow architecture. It discusses VSDN implementation and limitations in reducing the network events processed in the control plane.

4.3.1 Software-Defined Networking (SDN) Overview

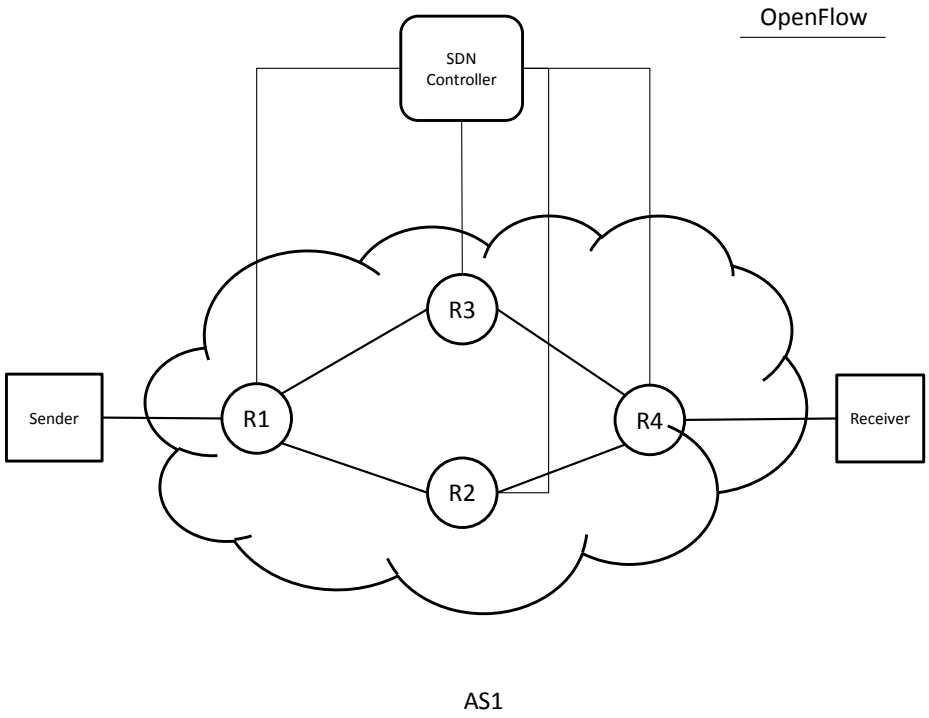

Figure 4.1.: SDN network with sender and receiver.

Software-Defined Networking (SDN) is a new approach in designing and developing computer networks [16]. SDN, using similar concepts as seen in server virtualization, allows computer networks to support rapid changing business needs. The key SDN concepts are abstractions—network as a graph, network virtualization, automation and orchestration of network services. SDN abstractions provide relevant information that applications use to improve their functionality and provide the means

for applications to specify desired behavior of network without the need to be aware of network configuration details [50].

SDN allows network applications and services to be rapidly developed and deployed. The control and data planes are decoupled and the network intelligence and state is relocated in a logically centralized controller. The network applications such as traffic engineering, network virtualization, and path resiliency are abstracted from the network and relocated in logically centralized controller as illustrated in Figure 4.1. The controller views network infrastructure including switches, routers, links and middle-boxes as a graph; this is the core of the SDN architecture. The SDN architecture exposes the flow tables of network infrastructure through an open programmable interfaces such as OpenFlow [21] or ForCES [67] which program the behavior of the network.

4.3.2 Video Over Software-Defined Networking (VSDN) Implementation

VSDN addresses rigidity of the path selection process of QoS network architectures [200]. VSDN provides end-to-end QoS guarantees for real-time interactive video applications. VSDN uses network-wide view to select optimum path for video applications using bandwidth, delay, and jitter. VSDN uses the SDN architecture and OpenFlow protocol to separate the control and data planes. The VSDN controller contains the routing logic and path selection application. The main element of VSDN architecture is the routing module (RM) that performs path computation.

VSDN Path Selection and Flow Installation

In Figure 4.1, when a request for network service is received from the sender, the switch—R1 forwards request using the default path—R1-R3-R4. Switch R4 forwards the request message to the receiver. The receiver generates an accept message and sends the accept message to R4. R4 forwards the accept message to controller. The controller uses the routing module (RM) [200] to find a feasible path that meets

path constraints—bandwidth, delay, and jitter. The RM returns a list of feasible paths to the controller. The admission control module processes the feasible paths to determine if network resources are available. If network resources are available, the controller installs the path—sends a modification message to each switch along the path—R4-R2-R1. After the path is installed, the controllers returns the accept message to R4 that forwards the accept message to R2. R2 forwards the accept message to R1. R1 forwards the accept message to sender.

VSDN installs optimum path, but generates one control plane event for each switch in the selected path. VSDN using explicit routing generates a single control plane event for installing an optimum path. For example, if the controller uses explicit routing the accept message is altered to include the path R4-R2-R1; the controller generates one control plane event instead of three events. The controller sends the accept message to R4 only.

The scalability of controller impacts deployment of SDN [82, 151, 158, 163, 260]. The scalability of the controller is improved by reducing the amount of state distributed from the control plane to data plane [149]. Explicit routing is needed to address the controller scalability problem. The remainder of this chapter discusses how Explicit Routing in Software-Defined Networking (ERSDN) addresses the controller scalability problem.

4.4 Design and Implementation

This section discusses design and implementation of Explicit Routing in Software-Defined Networking (ERSDN). This section discusses how ERSDN addresses challenge introduced in Section 4.3.2.

4.4.1 VSDN Flow Installation

Figure 4.1 represents a four-node SDN network. SDN/OpenFlow architecture, same as VSDN, installs flows hop-by-hop. In Figure 4.1, the receiver receives request

and responds with an accept message. The switch R4 receives accept message from receiver. The switch R4 checks its flow table for a matching flow. If flow does not match R4 forwards accept message to controller. The controller checks with policy control to ensure receiver can reserve network resources. The controller requests the routing module (RM) to find an optimum path such as R4-R2-R1. The controller checks with admission control to determine if network resources are available for the request. If network resources are available the controller installs the selected path. The controller starts with R4; then, installs flow in R2 followed by R1. In this example, one accept message generates three control plane events. Reducing network events such as accept messages in the control plane using explicit routing is the main idea of this chapter.

4.4.2 Design Choices

The following design choices were made to reduce the events in control plane:

1. For each VSDN message such as accept only signal edge router—R1 or R4, in Figure 4.1.
2. Allow core or access routers—R2 and R3 to install flow after receiving VSDN message from another switch.
3. Append required path to accept message, enabling each switch to process accept message and install required flow. This process is the same for other VSDN messages such as remove [200].

4.4.3 VSDN Purposed Flow Installation

In Figure 4.1, the controller receives the accept message from R4, the controller checks with policy control to ensure receiver can reserve network resources. The controller uses routing module (RM) to find an optimum path which meets the path constraints such as 3.0 Mbps, 60 ms delay, and 30 ms jitter. The path R4-R3-R1

meets path constraints. The controller appends the path to accept message. The controller configures flow in R4. The controller sends altered accept message to R4. R4 processes accept message and forwards the accept message to R3. The accept message is forwarded to R3 that processes and forwards the accept message to R1. R1 forwards accept message to sender, in Figure 4.1.

4.4.4 VSDN Switch Implementation Changes

Algorithm 1 Process Accept Message In Switch

```
procedure ProcessAcceptMessage(OFM)
    OFM: OpenFlow Modification

    if (FlowIsConfigured(OFM) == FALSE) then
        DoOutPut(OFM.BufferId, InPort)
    else
        if (SwitchType() == ACCESS) then
            AddVSDNShaper(OFM.OutPutPort, OFM)
        end if
        R = AddFlow(OFM)
        B = RetrieveBuffer(OFM.BufferID)
        ExecuteActions(OFM.BufferID, B, OFM.Actions)
    end if
end procedure
```

In algorithm 1, the changes to the switch are shown. In procedure ProcessAcceptMessage, the switch receives an OpenFlow Modification (OFM) that determines if flow is configured—FlowIsConfigured(OFM). If flow is configured, the switch passes buffer—packet out port to its destination—DoOutPut.

The flow is added—AddFlow if the flow is not configured. If the switch is an ACCESS—ingress or egress, the switch installs a traffic shaper using AddVSDNShaper. The traffic shaper ensures QoS of the flow. The buffer is retrieve using RetrieveBuffer. The ExecuteActions performs required actions on buffer such as output to switch port, set MPLS label, or vendor specific action.

The major change in switch is the ability of switch to add its flow. The ProcessAcceptMessage procedure can be called by the controller sending a modification message or switch receiving an accept message on its input port.

4.4.5 VSDN Controller Implementation Changes

In algorithm 2, the changes to controller are shown. The switch forwards the accept message to the controller. The procedure ProcessAcceptMessage in algorithm 2 receives the accept message as a buffer—B from switch network device—S. The controller retrieves video type using GetVideoTypeFromBuffer(B).

The GetContraintsFromVideoType retrieves video type constraints such as bandwidth, delay, and jitter. The policy control checks if request is over maximum allowable bandwidth—MaxBandwidthAllowed. The controller retrieves the network topology and determines ingress and egress switches—R1 and R4, in Figure 4.1. After the controller determines a set of feasible paths using GetFeasiblePath, it checks with admission control to determine if network resources can be acquired—AcquireFlowResources. If resources can be acquired, the path is appended to buffer—B using AppendPathToBuffer. The controller, after appending path to buffer, updates ingress switch R4 and returns the buffer to R4.

The major change in the controller is the appending of path to buffer before flow is installed in the switch. R4 forwards packet out port to next switch in path list which is R3. R1 removes the appended path from the packet before forwarding packet to the sender.

Algorithm 2 Process Accept Message in Controller

procedure PROCESSACCEPTMESSAGE(S, B)

S: OpenFlowSwitchNetDevice

B: OpenFlowBuffer

$V = GetVideoTypeFromBuffer(B)$

$EC = GetContraintsFromVideoType(V)$

if $(PolicyControl.MaxBandwidthAllowed(EC))$ **then**

return $FALSE$

end if

$G = TopologyMonitor.GetNetworkTopology()$

$R4 = GetIngressSwitch(G)$

$R1 = GetEgressSwitch(G)$

$P = GetFeasiblePath(R4, R1, EC)$

$R = FALSE$

if $(AdmissionControl.AcquireFlowResources(P, V))$ **then**

$AppendPathToBuffer(P, B)$

$UpdateIngressSwitch(R4, B)$

$R = TRUE$

end if

return R

end procedure

4.5 Results

This section analyzes the events generated in the control plane by VSDN and ERSDN.

4.5.1 Experimental Setup

Performance metrics - This chapter chooses following performance metric to assess the performance of ERSDN.

- *Network plane messages* - measures the messages processed by the network planes of VSDN and ERSDN.

The experiments were performed on an AMD Athlon X2 5400 system configured with Fedora 20 and 4GB RAM.

4.5.2 Experimental Results

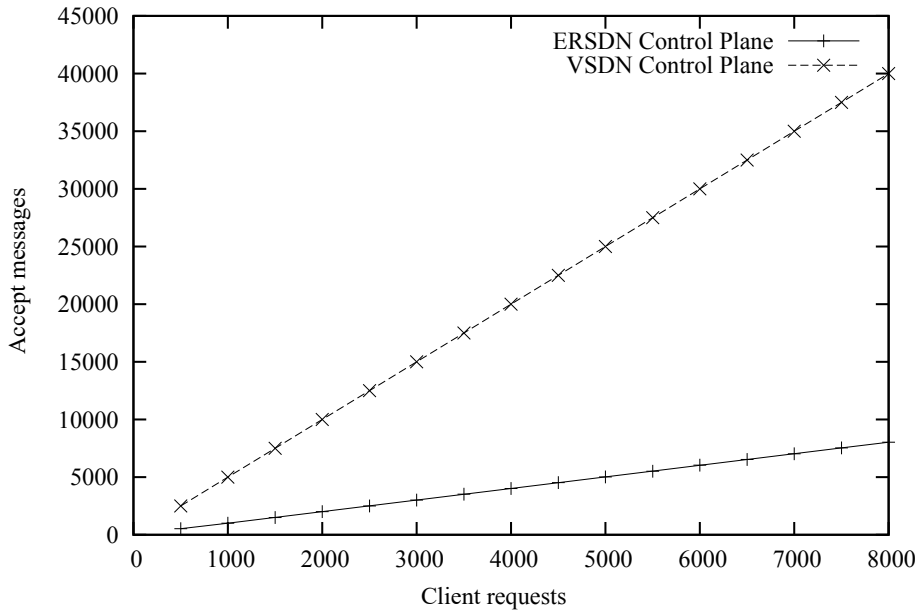

Figure 4.2.: The control plane accept messages generated for six-node network when client requests increase. ERSDN generates fewer accept messages compared to VSDN.

The accept messages in the control plane for VSDN are 2,000 at requests 500 in Figure 4.2. VSDN generates one accept message for each switch in the selected path which increases the accept messages in the control plane. ERSDN accept messages in the control plane are 500 at requests 500. ERSDN appends the path to accept message and forwards the accept message to the access switch only. Appending the path to accept message decreases the events in control plane in Figure 4.2. VSDN accept messages increase to 5,000 at requests 1,000 because one accept message is generated for each switch in the selected path. ERSDN accept messages are 1,000 at requests 1,000. ERSDN generates a single accept message which reduces the events in control plane. VSDN accept messages increase to 10,000 at requests 2,000 because VSDN generates an accept message for each switch along the path; each switch receives and forwards accept message to the controller. In Figure 4.2, the accept messages

grow faster for VSDN. ERSDN accept messages grow slower starting at requests 2,000. VSDN accept messages grow to 40,000 when client requests increase to 8,000. ERSDN accept messages grow to 8,000 when client requests increase to 8,000. There is an one-to-one ratio between control plane requests and accept messages using ERSDN. Although VSDN and ERSDN behaviors are linear, ERSDN grows slower.

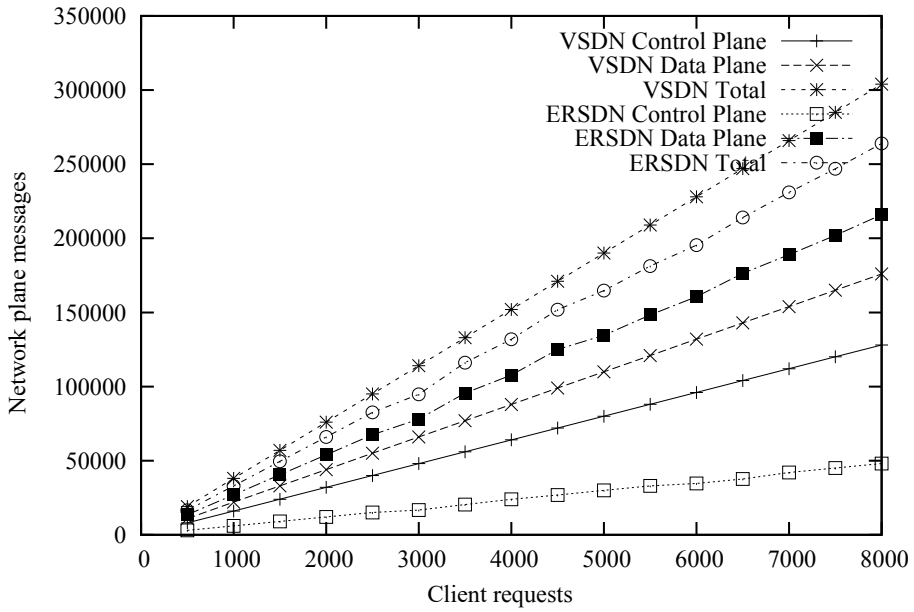

Figure 4.3.: The control plane messages generated for six-node network when client requests increase. ERSDN generates fewer messages compared to VSDN.

The messages for each network plane increase linearly when client requests increase in Figure 4.3. The messages in the control plane and data plane are similar with VSDN and ERSDN at requests 500. The control plane messages of ERSDN increase slightly from requests 500 to 1,000 because the accept messages increase with requests only and not the length of selected path. The messages increase noticeably from requests 500 to requests 1,000 because the accept messages are proportional to switches along the path using VSDN; the controller generates one network event for each switch along the path, increasing the events in control plane. The control plane messages increase at requests 2,000 using ERSDN. The control plane messages double using VSDN

because the controller generates one network event in control plane for each switch in the selected path at requests 2,000. When requests increase to 8,000, the events in control plane for ERSDN increase slower compared to VSDN. The control plane messages of VSDN are over 220,000, whereas the control plane messages of ERSDN are fewer than 50,000 at requests 8,000. ERSDN generates 260% fewer control plane events when client requests are 8,000.

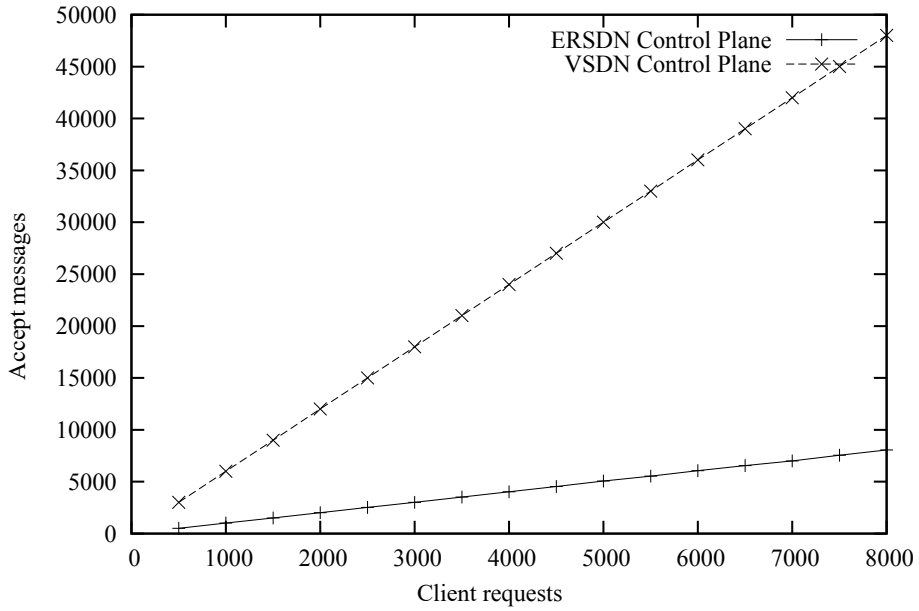

Figure 4.4.: The control plane accept messages generated for thirteen-node network when client requests increase. ERSDN generates fewer accept messages compared to VSDN.

The accept messages for VSDN are 3,500 at requests 500 in Figure 4.4. ERSDN accept messages are 500 at requests 500. ERSDN appends the path to accept message and forwards the accept message to access switch which decreases control plane events in Figure 4.4. VSDN accept messages increase to 6,000 at requests 1,000 because there is one accept message processed for each switch along the path. ERSDN accept messages are 1,000 at requests 1,000. ERSDN generates a single accept message which reduces control plane events. ERSDN accept messages are not affected by the

path length in Figure 4.2 and Figure 4.4. VSDN accept messages increase to 12,000 at requests 2,000 because VSDN generates an accept message for each switch along the path. In Figure 4.4, the accept messages increase using VSDN. ERSDN accept messages grow slower starting at requests 2,000. The VSDN accept messages grow to 48,000 at requests 8,000. The ERSDN accept messages increase to 8,000 at requests 8,000 because ERSDN sends one accept message to the access switch. VSDN and ERSDN message complexity are linear, but ERSDN increases slower.

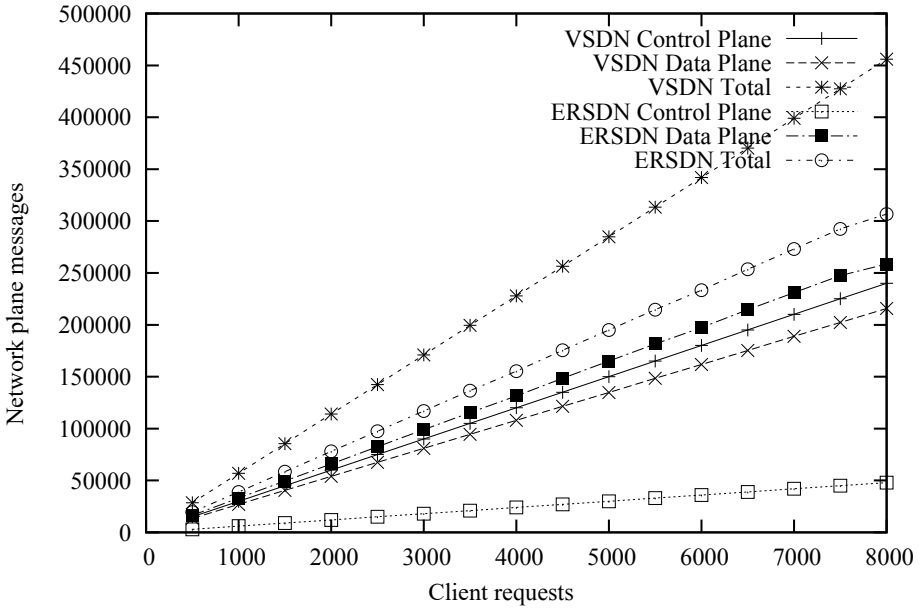

Figure 4.5.: The control plane messages generated for thirteen-node network when client requests increase. ERSDN generates fewer messages compared to VSDN

The messages in the control and data planes are linear in Figure 4.5; this behavior is similar to the behavior in Figure 4.3. The VSDN events in the control plane double at requests 1,000. The VSDN network events double at requests 2,000. The controller generates an event for each switch along the path, increasing the control plane events of VSDN in Figure 4.5. The ERSDN messages increase from requests 2,000 to requests 4,000. The VSDN messages double because of the switch-to-controller-processing of the accept messages at requests 4,000. As the requests increase, the ERSDN and

VSDN messages increase in Figure 4.5. The ERSDN messages are fewer than 50,000 at requests 8,000 because only the access switch receives accept message from the controller. These results in Figure 4.5 are similar to requests 8,000 in Figure 4.3 because the path length does not affect the events generated in the control plane using ERSDN. The network events are 216,000 for VSDN at requests 8,000. ERSDN decreases the control plane messages by 430% compared to VSDN at requests 8,000.

The explicit routing used by ERSDN decreases the events in control plane. ERSDN specifies the path in the accept message which avoids the need to perform flow installation for each switch in the path by the controller. The switch in ERSDN receives the accept message and installs flow. Explicit routing allows trade-off between time—control plane events and space—packet size due to appended path [314]. The design goal of ERSDN is to reduce the events in control plane. Reducing the events in control plane reduce stress on controller [248]. Source-based routing is needed to minimize state distribution [248].

4.6 Related Works

The authors [248] address controller scalability and performance issue by developing a new source-based routing scheme. The motivation, similar to research in this chapter, is to reduce state distribution between the controller and network infrastructure—switches to improve scalability and reduce network cost. The routing scheme [248] labels each network interface. The labeled interfaces are linked together to create a path that is a sequence of interfaces. The path is embedded in the packet at ingress router with a hop count that represents the position of packet in the path.

The routing scheme [248] pushes only information to one node. The approach in this chapter is similar to approach in [248] because the approach in this chapter only pushes state to the access router. Pushing state only to access router reduces state distribution required by the controller. The source packet headers [248] are added and removed by trusted egress nodes. This chapter differs from the approach

in [248] because the path is appended by SDN controller on receiving and processing the accept message. ERSDN is similar to approach in [248] because the path—source packet header is removed by the egress router. The state reduction [248] is similar to the results in this chapter; the results illustrate reduction in state distribution using source-based and explicit routing schemes. The SDN state reduction is an Internet draft [315] motivated to reduce burden on SDN controller. In the future, ERSDN could be standardized for real-time interactive applications such as video.

4.7 Conclusions

This chapter presented the design and implementation of Explicit Routing in Software-Defined Networking (ERSDN). ERSDN reduces the stress of the VSDN controller by reducing the events generated in control plane. A prototype of ERSDN was developed and the behavior of ERSDN was analyzed using the events generated in the control plane. ERSDN reduces the control plane messages generated by Video over Software-Defined Networking (VSDN) by 430% using an explicit routing scheme.

5 RELIABLE VIDEO OVER SOFTWARE-DEFINED NETWORKING (RVSDN)

5.1 Abstract

Ensuring end-to-end quality of service for video applications requires the network to choose feasible path using constraints such as bandwidth, delay and jitter. Quality of Service (QoS) can be ensured if the paths are *reliable*—perform to specification for each request. This chapter makes four contributions to the literature on providing end-to-end QoS for real-time interactive video applications. This chapter presents Reliable Video over Software-Defined Networking (RVSDN) that builds upon previous work—Video over Software-Defined Networking (VSDN) to address the issue of finding most reliable path of the network. This chapter presents the design and implementation of RVSDN. This chapter presents the experience of integrating RVSDN in ns-3, a network simulator used by the research community for simulating and modeling computer networks. This chapter presents the results of RVSDN and analyzes the results using requests serviced by the network. RVSDN services 31 times more requests than VSDN and MPLS explicit routing when reliability constraint is 0.995 or greater.

5.2 Introduction

Video traffic demands across the Internet are projected to be 69% of the Internet traffic by 2017 [316]. The increase in video demand is caused by hardware such as smart TVs, tablets, and smart phones and software such as Facebook, YouTube, Netflix, and HuLu. Real-time interactive video applications such as video-on-demand (VOD) and telesurgery are pushing the network infrastructure and protocols to the

limits. The real-time interactive video applications require specific level of Quality of Service (QoS) from the network. The QoS that the network provides to video applications can be bandwidth and minimal delay and jitter. The network QoS frameworks such as differentiated services, integrated services and Multiprotocol Label Switching (MPLS) provide limited QoS for real-time application such as video [311]. The network QoS frameworks and the Internet were not developed for the demands of real-time interactive video applications; therefore, new approaches for QoS, security, reliability, and wireless technologies within the Internet are needed [317].

Supporting real-time interactive video applications requires rethinking about how the network provides end-to-end QoS guarantees. The network QoS frameworks consider bandwidth, delay and jitter constraints. Although bandwidth, delay and jitter are important to real-time applications such as video, meeting the constraints do not address reliability of paths or build confidence of the network operator about path selection process. In this chapter, reliability is ability of network to perform to specification [308] for each request. This chapter builds on past work [200] and investigates ability of the network to select reliable end-to-end path. Network frameworks can use multi-path selections to decrease failure probability, respond to failures, or increase bandwidth capacity [4]. Multi-path selection allows the network to load balance network traffic and to provide failover in case the primary path fails. Multi-path selection does not address reliability requirement for video applications such as remote surgery, robotic packets, or interactive video.

Constraint-based routing or multiple path selection can fail to provide end-to-end QoS for real-time interactive video applications. For example, if QoS requirements for video application is 0.95 reliability, 1.5 Mbps bandwidth, 100ms delay, and 20ms jitter, the QoS frameworks such as integrated services find a Path1 that meets the constraints—bandwidth, delay and jitter, but are unable to meet reliability constraint because supporting reliability is not built-in the network design. The network operator may configure MLPS failover links to address issue of reliability. Although availability of network is increased using failover links, failover links may not pro-

vide required reliability for video application because MPLS explicit routing paths are static. If the network operator configures MPLS failover as Path1 with reliability of 0.75 and Path2 with reliability of 0.75, the overall reliability of the disjointed paths is $(1 - (1 - 0.75) \times (1 - 0.75)) = 0.9375$ which does not meet reliability of requirement—0.95. Furthermore, the static routes of MPLS cannot dynamically select a combination of reliable paths to service real-time applications such as interactive video.

Meeting reliability requirement for video applications requires the path selection process to consider more than a single path even if the path meets reliability requirement before failure. It requires path selection algorithms to dynamically consider combination of bandwidth, delay, jitter, and reliability constraints across multiple paths. The network QoS frameworks such as differentiated services, integrated services and MPLS do not address issue of reliability which is the main idea of this chapter.

Therefore, this chapter presents the experience and results of integrating reliability support into Video over Software-Defined Networking (VSDN), a network architecture that ensures end-to-end QoS for real-time interactive video applications. The main contributions to research are. This chapter presents Reliable Video over Software-Defined Networking (RVSDN), a network architecture that builds on previous work [200] to ensure end-to-end QoS for video application. This chapter presents results of implementing prototype of RVSDN. This chapter presents a study that evaluates the runtime performance of RVSDN using reliability.

Chapter organization. The remainder of this chapter is organized as follows. Section 5.3 motivates need for Reliable Video over Software-Defined Networking (RVSDN). Section 5.4 discusses the design and implementation of RVSDN. Section 5.5 presents and interprets simulation results. Section 5.6 compares RVSDN to related works. Section 5.7 provides concluding remarks.

5.3 Integrated Services (IntServ) and Video over Software-Defined Networking (VSDN)

This chapter discusses the limitations of IntServ and VSDN architectures in providing QoS for real-time interactive video applications.

5.3.1 Integrated Services (IntServ)

IntServ architecture uses a reservation protocol to configure end-to-end QoS over IP networks. The network resources such as memory, central processing unit (CPU), and bandwidth are reserved at each router along the path. The PATH message is sent by the sender to receiver. The PATH message follows the same path as IP packet; the PATH message cannot be sent using a different path. The receiver responds to sender with RESV message that reserves network resources along the path. The PATH and RESV messages configure soft states such as rate, max queue size, peak rate, and minimal packet size at each router along the path. These metrics ensure the packets belonging to specific flow receive guaranteed QoS. The soft states at each router are periodically refreshed to avoid session timeout.

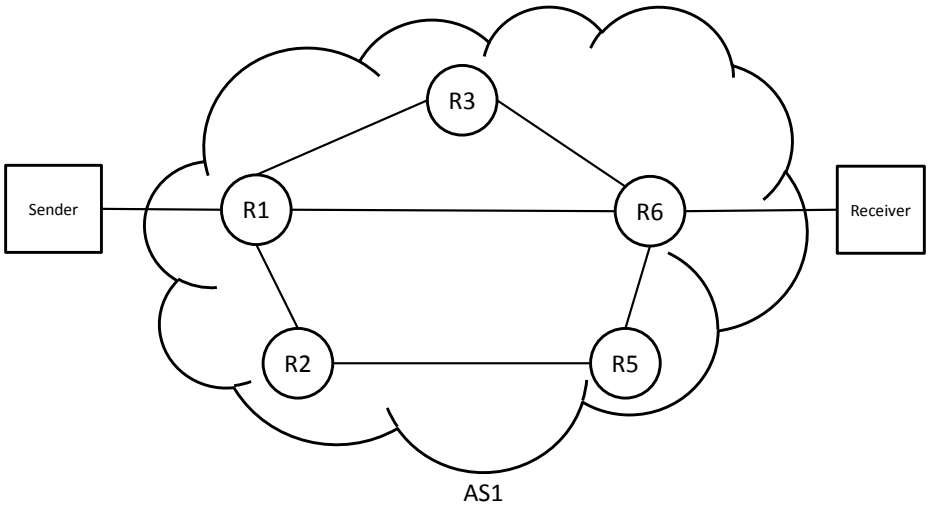

Figure 5.1.: A network with sender and receiver where routers in autonomous systems one (AS1) make independent decisions about path selections. Finding a reliable path across AS1 is difficult because each router makes its own routing decision.

IntServ advantages include software state adaptability, ability of receiver to initiate reservation, and ability for routers to merge reservations [311]. One disadvantage of IntServ is inability to select a different path from the routing protocols—Intermediate System to Intermediate System (IS-IS), Open Shortest Path First (OSPF), or Routing Information Protocol (RIP). For example, the best path for video using bandwidth, delay, and jitter constraints is R1-R2-R5-R6 in Figure 5.1. If routers in AS1 are running OSPF, the shortest path is R1-R6; therefore, video packets between the sender and receiver traverse path R1-R6 which is two hops, using IntServ.

In this example, IntServ ensures QoS at routers R1 and R6, but the best path is R1-R2-R5-R6 and not R1-R6. Furthermore, if path R1-R6 fails, OSPF finds the next shortest path that is R1-R3-R6. IntServ installs reservations along path R1-R3-R6 after, but the best path is R1-R2-R5-R6. IntServ has failed to configure QoS along the best path R1-R2-R5-R6.

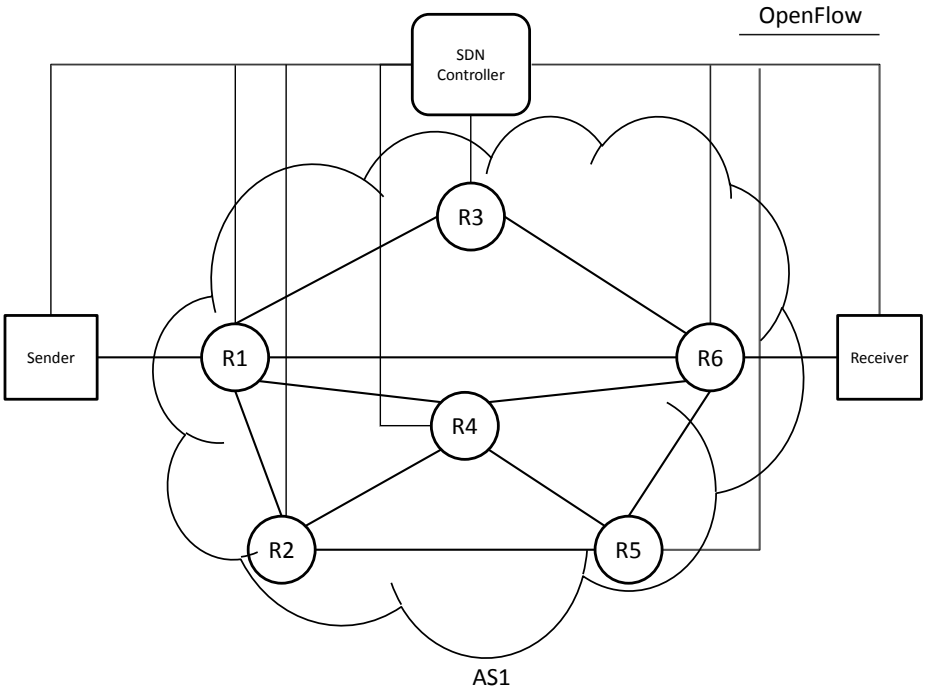

Figure 5.2.: Software-defined networking (SDN) network with sender and receiver. The controller programs the behavior of network including sender and receiver.

In Figure 5.2, VSDN architecture addresses the path inflexibility limitation of IntServ architecture. VSDN, similar to IntServ, provides QoS guarantees to real-time interactive applications such as video. VSDN selects optimum path for video application using bandwidth, delay, and jitter constraints.

5.3.2 VSDN Limitations

Although VSDN selects the optimum path for real-time interactive video application using bandwidth, delay and jitter, VSDN is limited in two ways when supporting end-to-end QoS for video applications such as telesurgery. VSDN considers a single path when servicing the request of video application. A single path may fail, negatively affecting the performance of video application. For example, in Figure 5.2, if packets traverse path R1-R6 and the path fails, VSDN recognizes the failures and finds a different path.

VSDN does not consider reliability when making path selections. VSDN could aggregate reliability over multiple paths to ensure QoS for real-time interactive video application. For example, in Figure 5.2, VSDN have to reject request or service the request with no guarantee using reliability of 0.92—if the paths have reliability of 0.92 and the video application requests reliability of 0.95. If VSDN aggregates reliability across two paths such as R1-R6 and R1-R3-R6, VSDN can ensure reliability of 0.9936—$(1 - (1 - 0.92) \times (1 - 0.92)))$ across multiple paths.

The network has multiple paths, mobile devices have multiple radio interfaces, computer devices have multiple network interfaces, and data centers have multiple paths [318]. The client of video application needs to support multiple path transport to take advantage of the reliability support of RVSDN. MultiPath TCP (MPTCP) [318] has been shown to be feasible. A detailed explanation of multi-path transport layer support is outside the scope of this chapter.

The network could select most reliable paths and aggregate reliability across multiple paths. Aggregating reliability across multiple paths allows the network to perform

to specification for each request. Having the network operator configure explicit paths across links using a protocol such as MPLS is *hard*. The remainder of this chapter discusses how Reliable Video over Software-Defined Networking (RVSDN) addresses issue of reliability.

5.4 Design and Implementation

This section discusses the design and implementation of Reliable Video over Software-Defined Networking (RVSDN). This section discusses how RVSDN addresses the challenges introduced in Section 5.3.2.

Video Over Software-Defined Networking (VSDN)

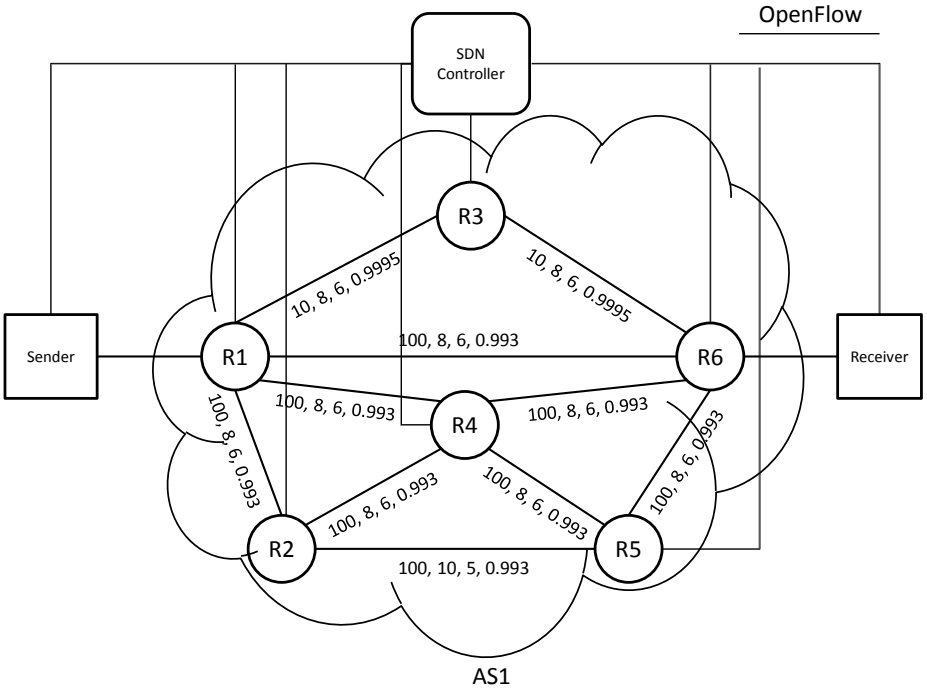

Figure 5.3.: SDN network with link constraints—bandwidth, delay, jitter, and reliability.

Figure 5.3 shows the architecture of VSDN. The links are labeled with QoS constraints—bandwidth Mbps, delay ms, jitter ms, and reliability. VSDN uses SDN [16] and OpenFlow [21] to separate the control and data planes of network devices—switches. The control plane—routing is implemented in the controller which resides outside of data plane—forwarding in Figure 5.3. The control plane and data plane communicate with one another using OpenFlow. A key component of VSDN is the Routing Module (RM) [200]. The RM is located in the controller and the RM uses constraint-based routing to calculate feasible paths [200]. Constraint-based routing (CBR) with two or more constraints is NP-hard [307]; therefore, a heuristics—A*Prune Algorithm finds feasible paths through the network.

A*Prune Algorithm

A*Prune algorithm combines A*Search with a pruning technique [319]. A*Prune algorithm solves finding K shortest paths subject to multiple constraints (KMCSP). A*Prune algorithm takes a graph G—vertices V, and edges E. A*Prune starts at path P(s, s) where s is a starting vertex in G. It expands paths P(s, V) that are reachable from s. A*Prune performs specific pruning against constraint C, only paths in admissible head path set P(s, V, H(p), C) are considered. The paths are ordered in a way that the path with shortest project path length $H_0(p)$ is expanded first. The algorithm terminates when there is a set of constraint shortest path (CSP) found or there are no candidate paths found. There are 7 key processing steps in A*Prune which are combined to select, expand, and prune candidate CSP until the algorithm terminates [319].

5.4.1 VSDN Routing Module Changes

VSDN Routing Module (RM) uses a variation of A*Prune algorithm [319] to perform constraint-based routing using bandwidth, delay, and jitter as metrics. RVSDN supports reliability constraint and aggregation of reliability across multiple paths

unlike VSDN. A path supporting real-time interactive video applications such as telesurgery may support bandwidth, jitter, and delay constraints, but if the path is unreliable, the performance of network for requests cannot be guaranteed. RVSDN provides QoS for real-time interactive video applications that require constraints such as bandwidth, delay, jitter, and reliability.

Algorithm 3 Find Reliable Paths

procedure FINDRELIABLEPATH(G, B, D, J, R)

G: Network Graph

B: Bandwidth

D: Delay

J: Jitter

R: Reliability

$EC = CreateEdgeConstraint(B, D, J, R)$

$R1 = GetIngressSwitch(G)$

$R6 = GetEgressSwitch(G)$

$P = GetFeasiblePath(R1, R6, EC)$

$RP = GetReliablePath(P, R)$

return RP

end procedure

Algorithm 3 illustrates pseudo code that is implemented in the controller to support reliable path selection. The user creates an edge constraint EC that takes parameters B, D, J, and R where B is bandwidth, D is delay, J is jitter, and R is minimal reliability for the video application. The ingress and egress switches—R1 and R6, in Figure 5.3, are retrieved using GetEgressSwitch and GetIngressSwitch. The GetFeasiblePath(R1, R6, EC) is a functionality of routing module (RM). The GetReliablePath(P, R) takes candidate paths P and reliability constraint R as pa-

Algorithm 4 Install Reliable Paths

procedure INSTALLRELIABLEPATH($RP, UUID$)

RP: Reliability Path(s)

$UUID$: Unique Path ID

if ($AcquireFlowResource(RP)$) **then**

$PathDatabase.Insert(UUID, RP)$

$InstallReliablePath(OFPFC\_ADD, RP)$

return $TRUE$

end if

return $FALSE$

end procedure

rameters. The GetReliablePath(P, R) sorts candidate paths. The RVSDN controller installs paths as illustrated in algorithm 4. The RVSDN controller acquires resources for reliability paths using AcquireFlowResource(RP). The reliability paths are stored in the path database using a unique id—UUID. The controller installs reliability paths using InstallReliablePath(OFPFC_ADD, RP).

For example, assume a video application requests reliability of 0.993 and FindReliablePath(G, B, J, R) returns 4 paths with reliability 0.91, 0.75, 0.94 and 0.89. The GetReliabilityPath(P, R) sorts paths P—0.94, 0.91, 0.89, 0.75. The GetReliabilityPath(P, R) determines if first path with reliability 0.94 meets reliability constraint. If not, it calculates reliability of first two paths—$0.94 + 0.91 - 0.94 \times 0.91$ which is 0.9946. A reliability of 0.9946 meets constraint for reliability 0.993. The GetReliablePath(P, R) returns reliability paths RP—paths with reliability of 0.94 and 0.91. The controller updates path database and flow tables of switch in the paths RP, after admission control—AcquireFlowResource(RP) determines that network resources are available for the request.

Algorithms 3 and 4 illustrate the ease of use for controller developers to find constraint-based paths and update switches. RVSDN is integrated into ns3 [313].

5.5 Results

This section analyzes requests serviced by the network architectures using reliability—ability of the network architecture to perform to specification for request.

5.5.1 Experimental Setup

Performance metrics - This chapter chooses following performance metric to assess performance of RVSDN.

- *The requests serviced by the network* - measures the requests serviced by network—VSDN, MPLS, and RVSDN. The actual constraints for each request are shown in Table 5.1.

The experiments were performed on an AMD Athlon X2 5400 system configured with Fedora 18, ns-3 v3.16 [320], and 4GB RAM.

5.5.2 Experimental Results

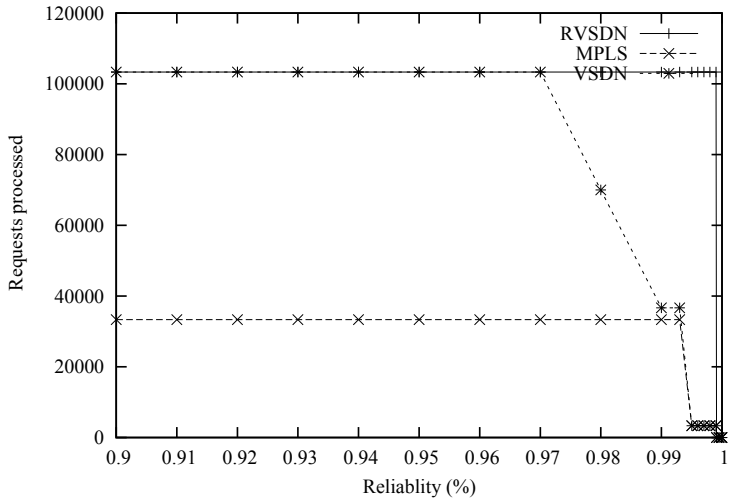

Figure 5.4.: The requests serviced by network architecture when video application reliability constraint increases. RVSDN services more requests than MPLS and VSDN because RVSDN aggregates reliability of multiple paths and dynamically discovers paths.

Figure 5.4 illustrates the requests serviced by the network architectures in Figure 5.3.

MPLS services 33,333 requests before rejecting other requests at reliability 0.90 because MPLS uses path R1-R6 which is 100Gbps. MPLS is using explicit routes—R1-R3-R6 and R1-R6 in Figure 5.3; therefore, after label switching paths resources are exhausted, MPLS is unable to guarantee QoS for video requests. VSDN and RVSDN

service 103,333 requests at reliability 0.90 because VSDN and RVSDN can explore undiscovered paths. RVSDN behaves similar to VSDN because there is no need to aggregate links at reliability 0.90. The paths in Figure 5.3 have reliability greater than 0.90 in Figure 5.4; therefore, as illustrated in Figure 5.4, each network architecture services requests between reliability 0.90 and 0.97.

MPLS services 33,333 requests at reliability 0.98. The path R1-R6 has reliability of 0.993 which meets reliability of 0.98 when using MPLS. The requests serviced by VSDN decreased to 70,000 requests at reliability 0.98 because VSDN does not aggregate reliability across multiple paths. VSDN uses reliability of a single path; therefore, only paths R1-R3-R6—$0.995 \times 0.995 = 0.990$, R1 - R6—0.993, and R1-R4-R6—$0.993 \times 0.993 = 0.9860$ meet reliability constraint of 0.98. RVSDN services 103,333 requests because RVSDN can aggregate reliability over multiple paths, at reliability constraint 0.98.

The requests serviced by VSDN decreases to 36,666 at reliability 0.99 because only paths—R1-R3-R6 and R1-R6 meet reliability constraint of 0.99. VSDN at reliability 0.99 behaves similar to MPLS because VSDN uses a single path when satisfying reliability constraint. MPLS services 33,333 requests at reliability constraint 0.99. The requests serviced by RVSDN remain constant at reliability 0.99 because it aggregates reliability across links. The requests serviced by each network architecture remain the same at reliability 0.993. The requests serviced by VSDN and MPLS drops to 3,333 because the architectures use path R1-R3-R6 that has bandwidth of 10Gbps and reliability of 0.999—$0.999 = 0.9995 \times 0.9995$. RVSDN continues to service 103,333 requests because RVSDN aggregates reliability across multiple paths which allows RVSDN to service more requests at reliability constraint 0.999.

The ability of RVSDN to aggregate reliability across multiple paths allows RVSDN to service more requests than MPLS and VSDN. VSDN services more requests than MPLS because the paths of VSDN are dynamically discovered and no explicit configuration is required by VSDN.

5.6 Related Works

The most reliable path (MRP) is determined by using a find shortest path first algorithm similar to Dijkstra or Floyd. Petrovic [321] uses labeling procedure and matrix algorithm to compute MRP. RVSDN uses a variation of the A*Prune algorithm with a combination of Dijkstra shortest path algorithm. RVSDN similar to [321] creates an adaptive routing process that selects most reliable path between nodes. RVSDN uses multiple network path aggregation to ensure reliability of service which differs from technique purposed by Petrovic [321].

Lee et al. [322] select most reliable path considering link cost and capacity such as average queue sizes. Lee et al. [322] use random early detection (RED), an algorithm for avoiding network congestion using buffer management in routers. Lee et al. [322] use Floyd shortest path algorithm and combines probability of packets dropped on link by RED algorithm to select MRP. RVSDN uses a variation of the A*Prune algorithm combined with Dijkstra shortest path algorithm to calculate MRP. RVSDN does not use queue length on links when calculating MRP. For RVSDN to support queue length, the switches need to report their average queue length to controller where the queue length can be used in MRP calculation. Statistic gathering is an expensive operation of OpenFlow switch [52]. More research is needed to determine if dynamic statistics gathering is cost effective. The idea of finding MRP under abnormal traffic conditions purposed by Lee et al. [322] is a technique that RVSDN could use to improve robustness.

Wang et al. [323] use MRP to ensure delivery of relief material after a natural disaster. Wang et al. [323] use concept of detour vital edge to choose adjustable reliable path that has higher connectivity reliability and minimal detour distant. Wang et al. [323] present three shortest path algorithms—depth first search and Dijkstra and modeled and compared modified versions of each algorithm. Wang et al. [323] use a modified version of Dijkstra shortest path to compute reliability and weight. Both traffic and communication networks can be complicated after a

natural disaster. Although networks are complicated, Wang et al. [323] illustrated feasibility and correctness of finding MRP after a natural disaster that RVSDN does not assume has occurred. RVSDN can use concept of detour vital edge and ability to function during abnormal traffic conditions [322] to calculate aggregated MRP in communication networks after a natural disaster. In this chapter, network failures were not introduced during experiment.

5.7 Conclusions

This chapter presented the design and implementation of Reliable Video over Software-Defined Networking (RVSDN). RVSDN builds on previous work [200] of providing end-to-end QoS for real-time interactive video applications that require bandwidth, delay, and jitter constraints. RVSDN added the support for reliability constraint to be used during the path selection process. RVSDN used multiple paths when determining if the network can service requests that require reliability. RVSDN services requests that required reliability of 0.999, whereas MPLS and VSDN ability to service requests decreased starting at reliability 0.995. RVSDN serviced 31 times more requests compared to VSDN and MPLS at reliability 0.995 or greater.

Table 5.1: The QoS constraints used during experiment where bandwidth, delay, and jitter remained constant, but reliability varied between 0.90 and 1.00.

Bandwidth	Delay	Jitter	Reliability
3.0 Mbps	150ms	30ms	0.90 - 1.00

6 MULTI-DOMAIN OVER SOFTWARE-DEFINED NETWORKING (MDVSDN)

6.1 Abstract

Supporting end-to-end quality of services for real-time interactive video applications such as videoconferencing and distance learning across the Internet requires a collection of independent networks or domains to work together to route packets between source and destination. Routing packets across the Internet using a feasible path—a path that meets the quality of service (QoS) attributes of video application is hard because the quality of service attributes such as bandwidth, delay, and jitter are not natively supported by the Border Gateway Protocol (BGP) which is the de facto routing protocol for the Internet. Although there may be multiple feasible paths between source and destination, BGP is unable to explore alternative paths because BGP advertises a single best path—feasible path, decreasing network flexibility.

This chapter provides three contributions to the literature on providing end-to-end QoS for real-time interactive video applications across the Internet. This chapter presents Multi-Domain Video over Software-Defined Networking (MDVSDN), a network architecture that selects end-to-end QoS path for real-time video application across independent domains. This chapter describes the architectural features of MDVSDN. This chapter presents results of implementing the prototype of MDVSDN and evaluates behavior of MDVSDN using message complexity. The message complexity of MDVSDN is linear.

6.2 Introduction

Supporting end-to-end quality of service (QoS) for real-time interactive video applications across the Internet requires a collection of independent network domains

to work together to provide QoS to the video applications such as videoconferencing and distance learning. The independent networks of the Internet work together to route packets between network devices that are geographically dispersed; therefore, performance of each network domain contributes to final service quality [324].

The QoS attributes such as bandwidth, delay, and jitter of network domains are not advertised by network operator and are not natively supported by Border Gateway Protocol (BGP) [325] which is the de facto routing protocol for the Internet. The real-time interactive video applications require guaranteed bandwidth, bounded delay [326], and bounded jitter [5], requiring the Internet to select a feasible path among multiple paths.

The Internet is vertically integrated where control plane—decision plane and data plane—forwarding plane are packaged together in network devices such as switches and middleboxes which makes it difficult to change the behavior of network. Due to tight coupling of the data plane and control plane, selecting a feasible path across the Internet is *difficult*. Furthermore, programming paths through the network is challenging because of closed interfaces of network devices. Software-defined networking (SDN) [16], a network architecture that decouples the data plane and control plane, has been proposed to address the challenges of managing network devices and programming the behavior of network. Nevertheless, the SDN solutions do not naturally support multi-domain environments [276].

Video over Software-Defined Networking (VSDN) [200], a network architecture that selects an optimum path among multiple paths, assumes the network domains operate independently of one another and path information is not exchanged between the network domains, thus decreasing the Internet's ability to provide end-to-end QoS for real-time interactive video applications. The network domains should coordinate flow setup originated by applications, containing information such as path requirement, QoS, and service-level agreement (SLA) across multiple SDN domains [327].

For example, there are two VSDN network domains, Domain1 and Domain2, peered. The sender is connected to Domain1 and the receiver is connected to Do-

main2. If sender initiates a video session with receiver, the sender sends receiver a request message. When receiver receives request message, the receiver generates an accept message. The accept message instructs Domain1 to install a feasible path which meets the video application QoS requirement. After installing path, Domain1 forwards accept message to Domain2 which configures a QoS path for video application. Domain1 and Domain2 independently install QoS paths without coordinating flow setup originated by video application. The two paths—Path1 that is selected by Domain1 and Path2 that is selected by Domain2 individually meet video application QoS requirement, but when stitched together may not meet QoS requirement such as delay and jitter of video application.

Because VSDN lacks the ability to select multiple domain QoS paths for real-time interactive video application, this chapter presents Multi-Domain Video over Software-Defined Networking (MDVSDN), a network architecture that selects end-to-end QoS multiple domain path for real-time interactive video applications.

The main contributions of this chapter to research are. This chapter presents Multi-Domain Video over Software Defined Networks (MDVSDN), a multi-domain network architecture that supports end-to-end QoS for real-time interactive video applications. This chapter presents results of implementing prototype of MDVSDN. This chapter analyzes MDVSDN behavior using message complexity.

Chapter organization The remainder of this chapter is organized as follows. Section 6.6 compares MDVSDN to related works. Section 6.3 motivates need for MDVSDN. Section 6.4 gives an overview of MDVSDN architecture and discusses implementation. Section 6.5 presents and interprets simulation results. Section 6.6 compares MDVSDN to related works. Section 6.7 draws together the topics discussed in this chapter.

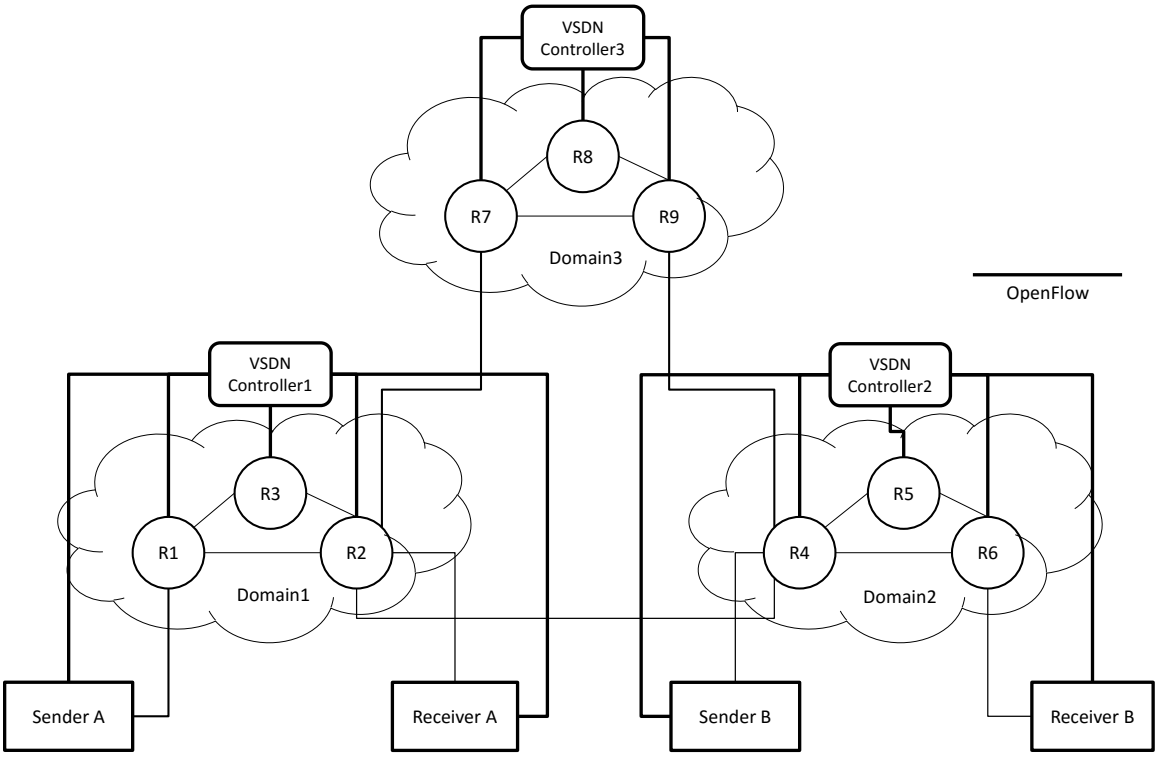

Figure 6.1.: VSDN network, illustrating three independent domains connected by links R2-R4, R4-R9, and R2-R7. The domains lack multi-domain flow management.

6.3 Motivation: VSDN Network

VSDN provides end-to-end quality of service for real-time video applications. VSDN uses the network-wide view to provision and select path that supports video application QoS requirements including bandwidth, delay, and jitter. VSDN supports three types—CIF, ED, and HD [200]. VSDN uses a variation of the A*prune algorithm to find Constrained Shortest Path First (CSPF) [307, 319].

VSDN simplifies application programmable interface (API) for video application developer, requiring minimal input to request service. VSDN has four message types, request—requests QoS from network, the video application generates a request message, accept—starts session, the receiver generates an accept message, remove—explicitly removes session from the network, the sender or receiver can generate a remove message, and error—indicates an error condition, the sender, receiver, controller, or switch can generate an error message. The sessions are implicitly removed by the network devices if flow timeout occurs [70].

VSDN operates on a single network domain. Although VSDN finds optimum path within a single network domain, it may not find optimum multi-domain path—paths across independent network domains that are stitched together to create an end-to-end path.

For example, in Figure 6.1, the best path for video application with respect to bandwidth, delay, and jitter is path Domain2-Domain3-Domain1 from sender A to receiver B. The VSDN controllers in Domain1, Domain2, and Domain 3 configure their paths independent of one other.

To initial session with receiver B, sender A sends a request message to R1, R1 forwards the request message to VSDN controller1. The controller determines sender A can reserve network resources within Domain1 and receiver B is unreachable from Domain1. The controller returns request message to R1 which forwards the request message over default path to R2. R2 forwards request message to R4—highest weight, local preference, or shortest path, in Figure 6.1. R4 forwards request message to

VSDN controller2. The controller in Domain2 determines receiver B is reachable from Domain2. The VSDN controller returns request message to R4 that forwards the request message through the default path to R6. R6 forwards request message to receiver B.

Upon receiving request, receiver B accepts the request from sender A by generating an accept message. The accept message is sent to R6. R6 forwards accept message to VSDN controller2. The controller performs admission control and policy control on request from receiver B. The controller in Domain2 calculates end-to-end path and installs flow rules and queues in receiver B, R6, and R4—path receiver B-R6-R4. After installing flow rules and queues, the controller returns accept message to R6. R6 forwards accept message to R4. R4 forwards accept message to R2. R2 forwards accept message to VSDN controller1. The controller receives accept message from R2 and performs admission control and policy control. The controller that manages Domain1 installs path sender A-R1-R3-R2 and returns accept message to R2. R2 forwards accept message to R3 and R3 forwards the accept message to R1 and R1 forwards accept message to sender A. The VSDN network has finished installing the multi-domain path between sender A and receiver B—path Domain1-Domain2; however, the best path for video application using bandwidth, delay, and jitter constraints is path Domain1-Domain3-Domain2.

The VSDN network is unable to select the best path for real-time video application across multi-domains because it lacks multi-domain flow management. The VSDN networks independently select the multi-domain path using local domain routing information. The multi-domain paths when stitched together may not be best path for real-time video applications.

One can attempt to solve this issue by having one VSDN controller program Domain1, Domain2 and Domain3 using slices [164]. Slices allow a single physical network to be used by multiple programs without harmful interference [164]. The network slices are static and are unable to be programmed dynamically [164].

Although VSDN networks select optimum path of a single domain, relying on VSDN networks to select feasible multi-domain path result in worst-case multi-domain path being selected. Selecting a feasible multi-domain paths requires two issues to be addressed.

Issue 1: Develop a network service layer that supports optimum multi-domain path selection among independent network domains. Selecting a multi-domain path across domains is feasible using a multi-domain network service layer that provides traffic engineering service. Selecting paths independently across domains, in Figure 6.1 may result in worst-case path being selected for real-time interactive video applications. A multi-domain network service layer with end-to-end visibility across domains needs to be developed.

Issue 2: Develop a control protocol that allows independent VSDN controllers to communicate with the network service layer and inform the network service layer of internal network state changes. A protocol that allows the controllers to request traffic engineering service from network service layer needs to be developed. The protocol should allow the controllers to inform the network service layer about internal state changes such as bandwidth, delay, and jitter [279, 328].

6.4 Architecture and Design

A network service layer in Figure 6.2 has been added to the VSDN architecture, addressing the issue in Section 6.3. The multi-domain SDN network architecture needs end-to-end visibility over the network to provide end-to-end QoS for video applications. For multi-domain SDN, the independent network domains are physically connected via the border switches [329]. MDVSDN uses a hierarchical design to increase scalability of network architecture [330]. Figure 6.2 illustrates a single MDVSDN controller for brevity. The network service layer can be made up of multiple

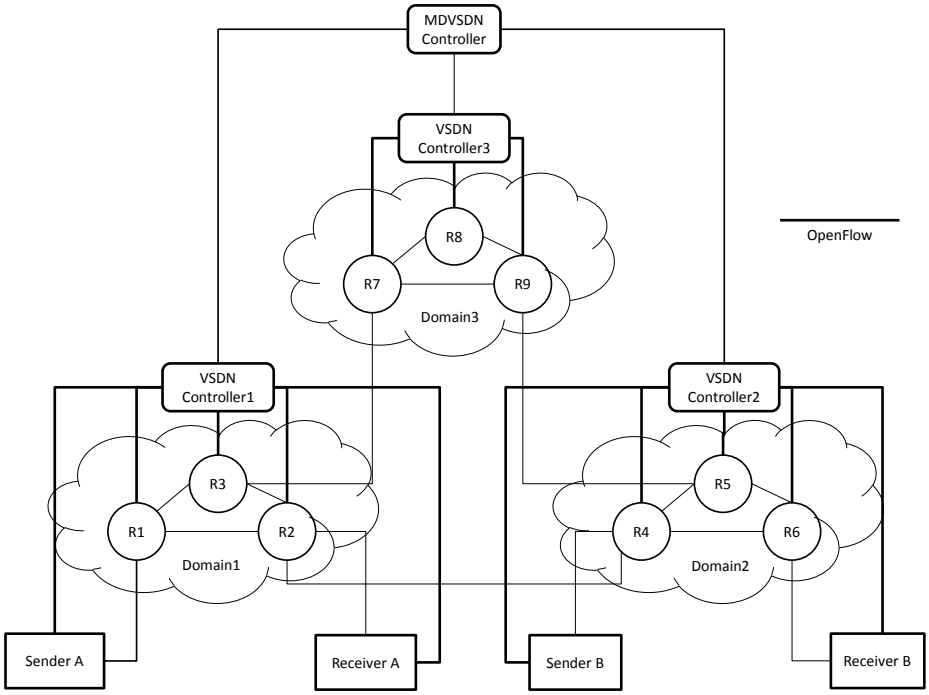

Figure 6.2.: MDVSDN network with three independent domains. Each VSDN controller has a view of its own network domain, lacking multi-domain flow management. The MDVSDN controller has the network-wide view which enables multi-domain flow management.

MDVSDN controllers that communicate with one another to establish an end-to-end path.

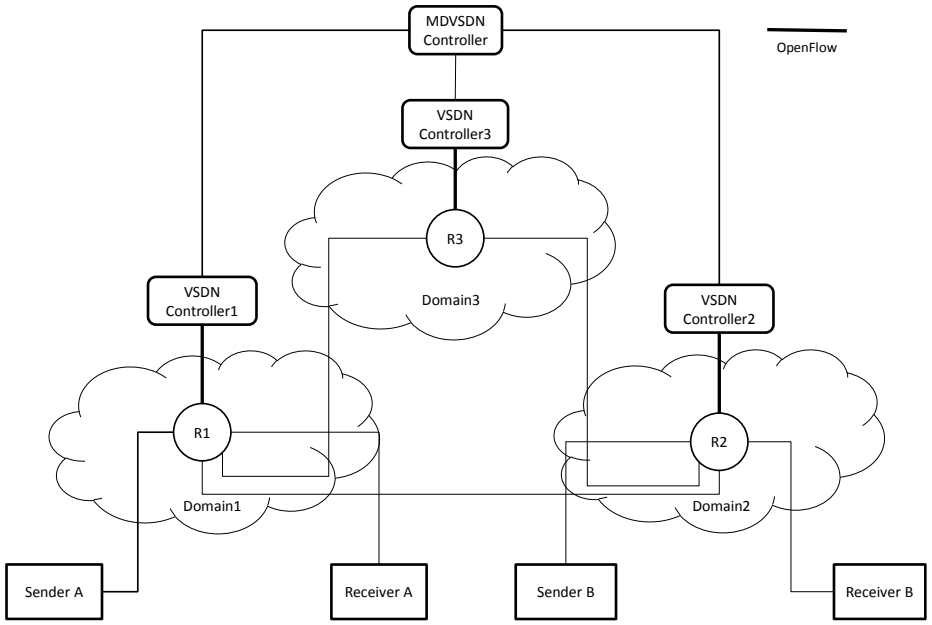

Figure 6.3.: A MDVSDN network view from the view of the MDVSDN controller. The MDVSDN controller does not have the local detail as the seen by the VSDN controllers. The MDVSDN controller sees an aggregated view of the network.

For the MDVSDN service layer to provide end-to-end QoS, each VSDN controller, in Figure 6.2, managed by the network service layer provides its aggregated network state to the network service layer. The aggregated state from controller includes average bandwidth, average delay, and average jitter. The aggregated state including reachability information and topology information is used by the MDVSDN controller to construct the network topology and to find feasible path for real-time interactive video applications, in Figure 6.3.

6.4.1 Controller

The service layer includes a global [82] or network-wide controller that communicates with the local [82] controllers to establish an end-to-end path and maintains

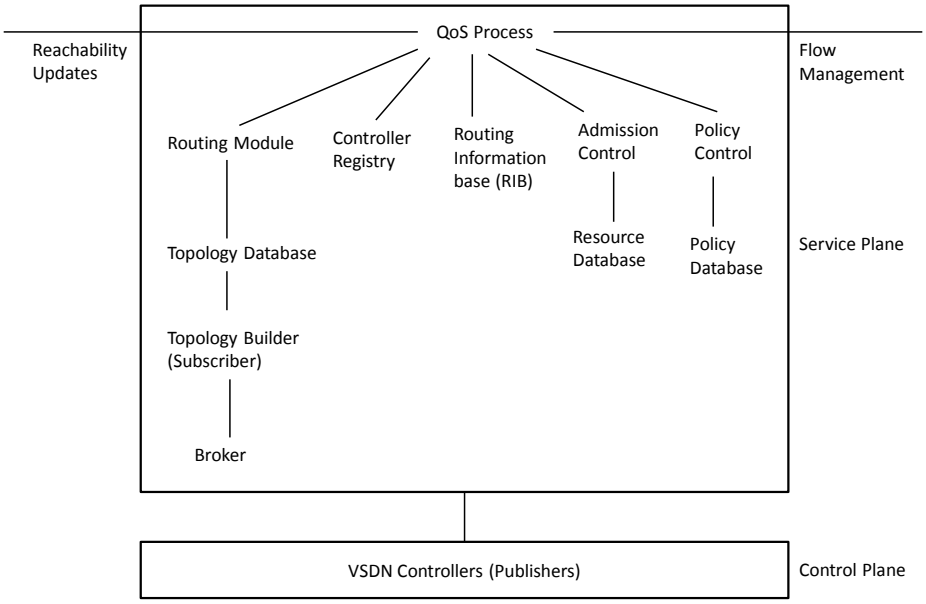

Figure 6.4.: MDVSDN controller, illustrating relationship between the elements. The MDVSDN controller services multi-domain QoS requests from VSDN controllers. The MDVSDN controller exchanges reachability information and QoS information with peering MDVSDN controllers, providing end-to-end multi-domain flow management and QoS.

global network resource database to capture the overall network—summarized topology and active flow information [278]. The MDVSDN controller receives aggregated topology information from VSDN controllers or publishers, reachability information from peering MDVSDN controller, and flow management information including QoS request from other independent MDVSDN controller [327]. The main elements of MDVSDN controller are shown in Figure 6.4.

In Figure 6.4, the controller registry stores the VSDN controllers that have registered with the MDVSDN controller. To request QoS, the VSDN controller registers with the MDVSDN controller using VSDN register message.

The routing updates received from other MDVSDN controllers or peers are stored in the routing information base (RIB). After routing process [110, 250] processes the updates using defined routing policies, the routes are stored in RIB.

The admission control, in Figure 6.4, maintains network status of paths including bandwidth. The admission control gives the MDVSDN controller an idea about status of network resources.

In Figure 6.4, the policy control provides consistency among VSDN controllers. The policy control contains information about communication policies among network domains. The text-based policies of network operator are converted into network configuration [205] using a policy translator. The policy control enforces security constraints that are configured by the network operator [284] and agreed upon by peering networks [331].

In Figure 6.5, the MDVSDN controller subscribes to topology updates from registered VSDN controllers, allowing the MDVSDN controller to build an aggregated view of network. The topology builder, in Figure 6.4, constructs the network topology using the topology updates from the VSDN controllers. The broker that the MDVSDN controller subscribes to decouples MDVSDN controller and VSDN controllers in time, space, and synchronization, thus increasing scalability of MDVSDN.

The routing module is used by the QoS process for finding feasible paths. The routing module paths may not be the optimum path because the network topology is

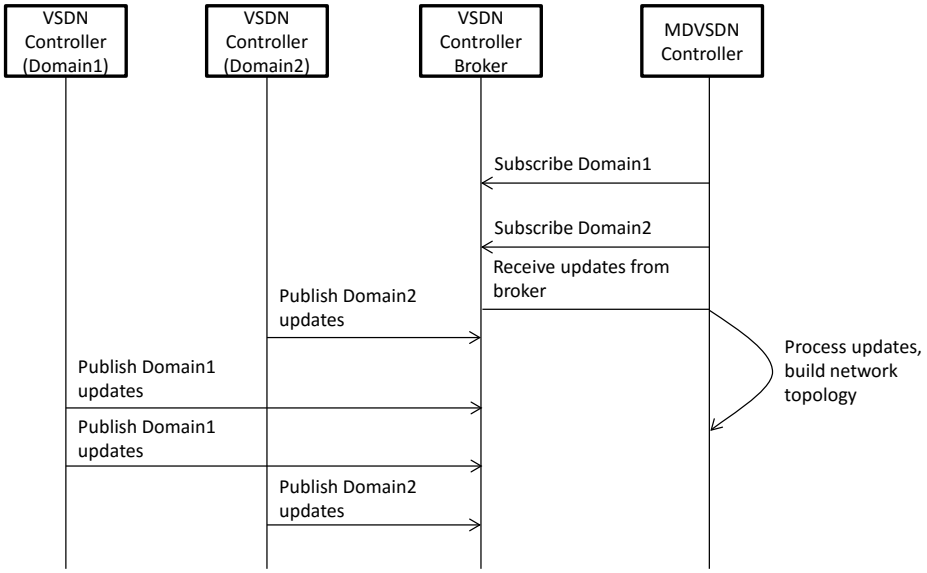

Figure 6.5.: VSDN publish-subscribe interaction diagram, illustrating how MDVSDN controller subscribes to topology updates.

an aggregated representation of the actual network. The path is a feasible path—path that meets video application QoS requirement. The routing module uses heuristics presented in [330] to identify feasible paths where source of inaccuracies is aggregation process that occurs in hierarchically interconnected networks [330]. The RM returns a list of subgraphs or paths that meet QoS constraints such as bandwidth, jitter, and delay [200].

6.4.2 Controller to Controller Communication

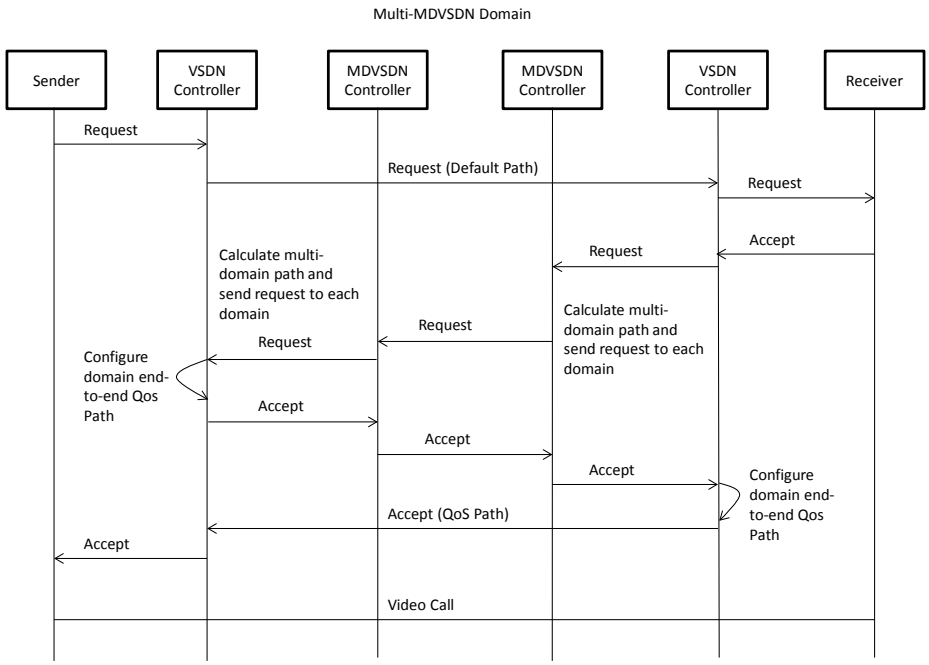

Figure 6.6.: MDVSDN controller communication, illustrating how independent MD-VSDN controllers communicate to establish a multi-domain end-to-end path. The VSDN controller and MDVSDN controller use the same protocol and messages which simplifies VSDN protocol and design.

The MDVSDN controllers communicate with one another to establish an end-to-end path over the Internet where there are tens of thousands of independent domains. When the MDVSDN controller receives a request that destination prefix is outside of

its domain, the MDVSDN controller looks up the prefix in its RIB. The MDVSDN controller calculates a path to next MDVSDN controller that is in the path of the destination and sends next hop MDVSDN controller VSDN request message. The next hop MDVSDN controller looks at the destination prefix and determines if the prefix is reachable. If destination is reachable, the MDVSDN controller calculates a feasible path and requests the VSDN controllers to configure a feasible path within their domains.

After the VSDN controllers configure feasible path, they send an accept message to the MDVSDN controller. The MDVSDN controller, after receiving the accept messages from the VSDN controllers, returns an accept message to source MDVSDN controller. The source MDVSDN controller calculates feasible path through the VSDN controllers, sends the VSDN controllers request message, after receiving the accept messages from the VSDN controllers, the MDVSDN controller returns the accept message to the VSDN controller that initially requested service. The VSDN controller that requested multi-domain network service configures a feasible path through its network domain, completing multi-domain path. The accept message is sent to sender. Figure 6.6 illustrates communication between MDVSDN controllers.

6.5 Simulation Results

Figure 6.7 illustrates VSDN messages generated by MDVSDN with two VSDN domains when network clients request service from the network. The VSDN message is 400 when the network client requests are 50. The vsdn-request and vsdn-accept messages equal 400 because each local VSDN controller processes packets that destination is local to its network. The mdvsdn-request and mdvsdn-accept message count equals 100 because there are two local controller domain managed by MDVSDN controller. Each VSDN domain controller sends inter-domain traffic request to MDVSDN controller, increasing the request and accept messages in network service layer—MDVSDN controller. The VSDN message count continues to increase as

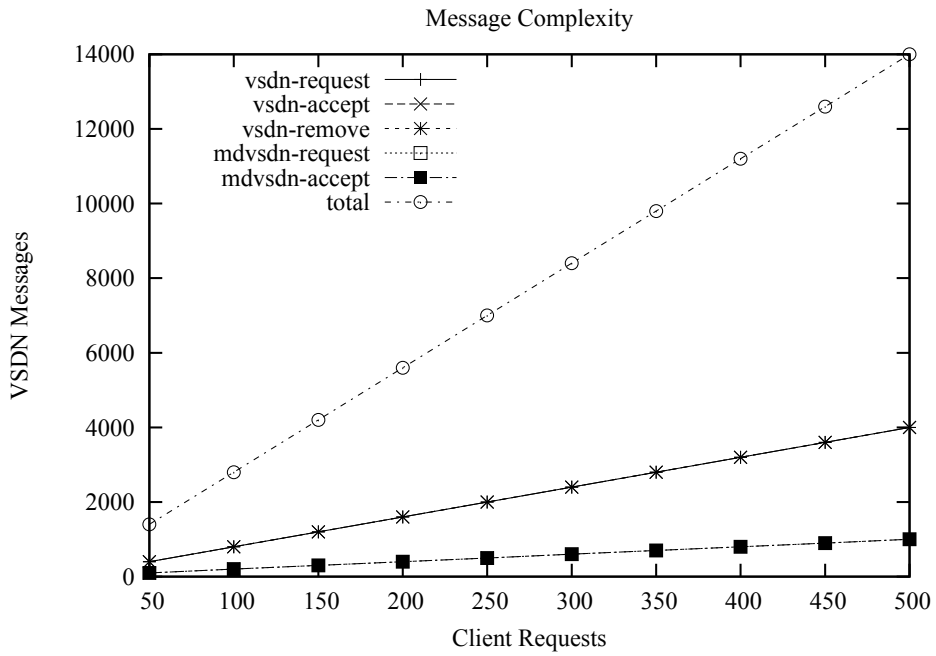

Figure 6.7.: Average VSDN messages generated when client requests increase. MD-VSDN message complexity is linear. There were two independent domains, two senders, and two receivers in simulation.

network client requests increases, in Figure 6.7. The highest VSDN message count is 14,000 when the network client requests are 500. The message complexity of MDVSDN is linear, in Figure 6.7. The MDVSDN controller messages are one-fourth the VSDN messages generated by the VSDN controller.

6.6 Related Works

Distributed Virtual Network Operations Center (DvNOC) [275] is a multi-domain SDN network framework [276] which combines SDN operations and management across multiple domains, integrating SDN and non-SDN environments. DvNOC incorporate three functionalities—multi-domain network awareness, efficient NOC-to-NOC cooperation, and user-oriented virtual network management. MDVSDN unlike DvNOC, which focuses on NOC-to-NOC collaboration, is specific to real-time interactive video applications across multi-domain SDN [276]. DvNOC uses east-westbound API to communicate between controllers in independent domains. The MDVSDN controllers—domain level communicate indirectly with one another using the service layer. The domain level controllers register with MDVSDN service layer that instructs the local controllers to configure end-to-end paths. MDVSDN service layer coordinates the multi-domain flow management.

The author [125] proposes a new inter-network paradigm that maintains heterogeneous Internet with independent domains, but incorporates market-driven multiple broker services between independent administrative domains [125]. The market-driven brokers in the control plane drive evolution and improvements during policy negotiation. The policy agreement between network domains is driven through market-driven incentives. MDVSDN avoids providing financial incentives for delivering video over multi-domain SDN. This chapter focuses on solving the technological challenges including multi-domain flow management using MDVSDN. MDVSDN and solution [125] are similar because both solutions provide a choice of end-to-end services using customer requirements, but differs because customer service varies between

service providers-brokers [125]. MDVSDN has four services tiers that the customer can choose. Service providers and independent domain that agrees to support VSDN services provide the same tier service to customer. The solution [125] may be used by MDVSDN for providing better QoS in each tier, thus using cognitive brokers with machine learning to route more intelligently.

Distributed Multi-domain SDN Controllers (DISCO) [60] is an open and extensible distributed control plan which manages Wide-Area Networks (WANs) such as the Internet. Each domain has its own SDN controller that manages its own network resources. The independent SDN controllers communicate with one another using a light-weight control channel [61]. The SDN controllers build a network-wide topology by sharing network state across control channels. DISCO provides failure recovery for inter-domain disruptions and end-to-end priority service requests, and supports virtual machine migration. Although MDVSDN and DISCO are similar in the service they provide, their architecture differs. The independent controllers of MDVSDN communicate indirectly through the service layer—brokers are responsible for finding end-to-end path for QoS request. Unlike Disco, MDVSDN local controllers communicate with one another using the service layer. MDVSDN uses a hierarchy architecture that allows local controller to control its network resources similar to DISCO. MDVSDN is design to support real-time interactive video applications such as Google Hangouts [332] and Skype [333].

B4 is a private WAN that connects the data centers of Google across the globe [273]. B4 main objective is to increase bandwidth utilization between data centers, where large scale data copies are performed. The applications that data traverses B4 network are prioritized, allowing higher priority applications to use more of bandwidth—dynamically allocating bandwidth as needed. B4 [273] uses SDN to centralized traffic engineering between data centers. MDVSDN is similar to B4 because the requirements of application drives adaptability of network—network adapts to QoS need of the application. MDVSDN differs from B4 because MDVSDN does not adapt network bandwidth with the application demands. MDVSDN guarantees QoS

to video application. The request is rejected if MDVSDN is unable to fulfill the request [2].

Video over Software-Defined Networking (VSDN) [200] is a network architecture which provides end-to-end QoS for real-time interactive video applications such as videoconferencing and distance learning. VSDN selects the optimum path among multiple paths [200]. The VSDN is a protocol that allows video applications to request service from the network. VSDN uses a tiered service model that allows video applications to request three levels of services—CIF, ED, and HD. MDVSDN builds on the idea of VSDN, introducing hierarchical VSDN network to improve the scalability of network. VSDN assumes a single controller for network which is infeasible for the Internet. Furthermore, independent VSDN controllers lack multi-domain network-wide view. The VSDN controllers are unable to locate the feasible multi-domain path. Similar to using independent VSDN controllers, MDVSDN adheres to video application request such as HD, but may not find the optimal path, an inherent behavior of hierarchical network architectures [330].

6.7 Conclusions

This chapter presented Multi-Domain Video over Software-Defined Networking (MDVSDN), a network architecture that provides end-to-end QoS for real-time interactive video applications across independent domains. This chapter describes the architectural features of MDVSDN. A prototype of MDVSDN was implemented and its behavior was analyzed using message complexity. The message complexity of MDVSDN is linear. MDVSDN selects feasible multi-domain path for real-time interactive video applications, improving the QoS and performance of video application across independent VSDN domains.

7 CONCLUSIONS

"The internal topologies of many networks are such that multiple paths can be found between most points. A major limitation of conventional IP forwarding is that single-metric, shortest-path trees use only one of the possible paths towards any given destination." Grenville Armitage

This thesis developed a network architecture that provides QoS for real-time interactive video applications such as videoconferencing, distant learning, and telesurgery.

In Chapter 1, this thesis outlined the requirements for a network architecture that supports real-time interactive video applications. The requirements were:

- The network applications and services needed centralized control of network resources.
- The network applications and services needed ability to program the behavior of network.
- The network needed to reject requests that the network is unable to service.
- The network needed to perform constraint based routing using bandwidth, delay, jitter, and reliability.
- The network needed to know traffic rates in advance.
- The network needed to enforce network policies consistently.
- The network architecture needed to support multi-domain end-to-end QoS path selection.

The network architectures such as DiffServ, IntServ, and MPLS were unable to meet requirements of the proposed network architecture. The network architectures had limitations such as inability to reject flows and select a feasible end-to-end path among multiple destination paths; therefore, the primary thesis research question was:

> What is the network architecture needed to support real-time interactive video applications?

In Chapter 2, this thesis summarized SDN research and presented taxonomy of SDN research. Chapter 2 gave motivation of this work and related the proposed network architecture to previous work on SDN. This thesis builds on SDN to develop a network architecture that determines how traffic flows through the network.

In Chapter 3, this thesis presented a network architecture and resource provisioning protocol—Video over Software-Defined Networking (VSDN) that selected a feasible constrained path among multiple paths to support real-time interactive video applications. VSDN used SDN network-wide view to manage network resources such as bandwidth. The message complexity of VSDN was linear.

In Chapter 4, ERSDN addressed VSDN controller scalability issue by reducing network events processed in the control plane by 430%. In Chapter 5, RVSDN addressed issue of finding the most reliable path for real-time interactive video applications such as telesurgery which requires reliability, bandwidth, delay, and jitter. RVSDN serviced 31 times more requests than VSDN and MPLS explicit routing when reliability constraint was 0.995 or greater. In Chapter 6, MDVSDN addressed the issue of selecting a feasible end-to-end path across independent domains. The message complexity of MDVSDN was linear.

The network architecture needed to support real-time interactive video applications has a logically centralized control plane that makes decisions using the network-wide view. The network architecture allows network applications and services to program the behavior of network. The network architecture knows network traffic characteristics in advance. The network architecture has linear message complexity.

The network architecture has an API that accepts three input parameters. The network architecture has a traffic engineering service that selects feasible multi-domain paths using bandwidth, delay, jitter, and reliability. The network architecture rejects requests that the network is unable to service.

7.1 Future Work

This thesis identifies five areas of future research.

- Implement VSDN in testbed and analyze how VSDN responds to changes in video application workload and traffic patterns
- Add ability to MDVSDN for selecting trusted paths such as routing traffic around certain locations during bad weather or where physical security is a concern
- Abstract and convert VSDN routing module (RM) to a TE service where other applications can request path selection service such as path selection using proximity to improve performance of VM migration and content delivery networks
- Extend VSDN prototype to support real world applications such as Google Hangouts and Microsoft Skype, illustrating how VSDN can improve performance of real-time interactive video applications
- Integrate VSDN into Mininet network emulator that creates network of virtual hosts, switches, controllers, and links on a laptop or personal computer

LIST OF REFERENCES

LIST OF REFERENCES

[1] Cisco. Cisco visual networking index: Forecast and methodology, 20142019. `http://www.cisco.com`, May 2015. White paper.

[2] Lawrence G. Roberts. The next generation of ip-flow routing. In *Proceedings of the International Conference on Advances in Infrastructure for e-Business, e-Education, e-Science and e-Medicine on the Internet*, 2003.

[3] Xipeng Xiao and Lionel M. Ni. Internet qos: A big picture. *IEEE Network*, 13:8–18, 1999.

[4] Shuchita Upadhyaya and Gaytri Devi. Mingling multipath routing with quality of service. *International Journal of Computer Science Issues (IJCSI)*, 8(5):156 – 161, Sep 2011.

[5] Grenville Armitage. *Quality of Service in IP Networks: Foundations for a Multi-service Internet*. Macmillan Publishing Co., Inc., Indianapolis, IN, USA, 2000.

[6] David McDysan. *QoS and traffic management in IP and ATM networks*. McGraw-Hill, Inc, New York, NY, USA, 2000.

[7] Vilho Raisanen. *Implementing Service Quality in IP Networks*. John Wiley & Sons, Inc., West Sussex, England, 2003.

[8] Eric D. Siegel. *Designing Quality of Service Solutions for the Enterprise*. Wiley Computer Publishing, Hoboken, NJ, USA, 1999.

[9] David Durham and Raj Yavatkar. *Inside the Internet's Resource Reservation Protocol: Foundations for Quality of Service*. John Wiley & Sons, Inc., New York, NY, USA, 1999.

[10] Z. Wang and J. Crowcroft. Quality-of-service routing for supporting multimedia applications. *Selected Areas in Communications, IEEE Journal on*, 14(7):1228–1234, Sep 1996.

[11] D.O. Awduche. Mpls and traffic engineering in ip networks. *Communications Magazine, IEEE*, 37(12):42–47, Dec 1999.

[12] Roberto Sabella and Paola Iovanna. Traffic engineering in next generation multilayer networks based on the gmpls paradigm. In *Ilkley, West Yorkshire, United Kingdom*, pages 26–28, Washington, DC, USA, 2004. IEEE.

[13] Ian Foster, Markus Fidler, Alain Roy, Volker Sander, and Linda Winkler. End-to-end quality of service for high-end applications. *Computer Communications*, 27(14):1375 – 1388, 2004. Network Support for Grid Computing.

[14] Sebastian Rampfl. Network simulation and its limitations, 2013. Seminar Future Internet SS2013, Lehrstuhl Netzarchitekturen und Netzdienste, Fakultt fr Informatik, Technische Universitt Mnchen.

[15] Jerry Banks, John S. Carson, Barry L. Nelson, and David M. Nicol. *Discrete-Event System Simulation*. Prentice Hall, Upper Saddle River, NJ, USA, 5th edition, 2010.

[16] Open Networking Foundation (ONF). Software-defined networking: The new norm for networks. `https://www.opennetworking.org`, 2012.

[17] Siamak Azodolmolky. *Software Defined Networking with OpenFlow*. Packt Publishing, Oct 2013.

[18] Fei Hu. *Network Innovation through OpenFlow and SDN: Principles and Design*. CRC Press, Feb 2014.

[19] Thomas D. Nadeau and Ken Gray. *SDN: Software Defined Networks*. O'Reilly Media, Aug 2013.

[20] Rajesh Kumar Sundararajan. *Software Defined Networking (SDN) - a definitive guide*. Rajesh Kumar Sundararajan, Jun 2013.

[21] OpenFlow. Openflow switch specification version 1.5.0. `https://www.opennetworking.org`, 2014.

[22] Antonio Manzalini, Roberto Saracco, Cagatay Buyukkoc, Prosper Chemouil, Slawomir Kuklinski, Andreas Gladisch, Masaki Fukui, Wenyu Shen, Eliezer Dekel, David Soldani, Mehmet Ulema, Walter Cerroni, Franco Callegati, Giovanni Schembra, Vincenzo Riccobene, Carmen Mas Machuca, Alex Galis, and Julius Mueller. Software-defined networks for future networks and services: Main technical challenges and business implications. White paper, IEEE, Nov 2014. 2013 Software Defined Networks for Future Networks and Services (SDN4FNS).

[23] Wenfeng Xia, Tina Tsou, Diego R. Lopez, Qiong Sun, Felix Lu, and Haiyong Xie. A software defined approach to unified ipv6 transition. In *Proceedings of the ACM SIGCOMM 2013 conference on SIGCOMM*, SIGCOMM '13, pages 547–548, New York, NY, USA, 2013. ACM.

[24] HP. Software defined networking: Create an intelligent, programmable, centrally-controlled network to master diverse applications and workloads. `https://www.hpe.com/us/en/networking/sdn.html`, 2015.

[25] NEC. Nec sdn solutions: Dynamic solutions for new business creation. `http://www.nec.com/en/global/solutions/sdn`, 2015.

[26] Big Switch. Sdn products: Modern network architecture enabled by bare metal fabrics. `http://www.bigswitch.com/products`, 2015.

[27] Arista. Software driven cloud networking. `https://www.arista.com/en/products/software-driven-cloud-networking`, Nov 2015.

[28] Elisa Bellagamba, Attila Takacs, and Joe Wilke. Software-defined networking: the service provider perspective. `http://www.ericsson.com`, Feb 2013.

[29] Ericsson. Service provider sdn. `http://www.ericsson.com`, Nov 2015.

[30] Juniper Networks. Software defined networking (sdn). `http://www.juniper.net`, Sep 2015.

[31] IBM. Ibm software defined networking, Sep 2015.

[32] Extreme Networks. Software-defined networking (sdn). `http://www.extremenetworks.com`, 2015.

[33] Nick Feamster, Jennifer Rexford, and Ellen Zegura. The road to sdn: An intellectual history of programmable networks. *Queue*, 11(12):20:20–20:40, 2013.

[34] O. Martikainen, J. Lipiäinen, and K. Molin. *Tutorial on Intelligent Networks*. Raportti / Lappeenrannan teknillinen korkeakoulu, tietotekniikan osasto. Lappeenranta University of Technology, 1994.

[35] David L. Tennenhouse and David J. Wetherall. Towards an active network architecture. *Computer Communication Review*, 26:5–18, 1996.

[36] D. Ruffen, T. Len, and J. Yanacek. Cabletron's securefast vlan operational model, 1999.

[37] Seong Gon Choi, Hyun Joo Kang, and Jun Kyun Choi. An efficient handover mechanism using the general switch management protocol on a multi-protocol label switching network. *ETRI Journal*, 25(5):369–378, Oct 2003.

[38] Th. Magedanz and F. C. de Gouveia. Ims – the ip multimedia system as ngn service delivery platform. *e & i Elektrotechnik und Informationstechnik*, 123:271–276, 2006.

[39] Matthew Caesar, Donald Caldwell, Nick Feamster, Jennifer Rexford, Aman Shaikh, and Jacobus van der Merwe. Design and implementation of a routing control platform. In *Proc. Networked Systems Design and Implementation*, Berkeley, CA, USA, 2005. USENIX Association.

[40] Albert Greenberg, Gisli Hjalmtysson, David A. Maltz, Andy Myers, Jennifer Rexford, Geoffrey Xie, Hong Yan, Jibin Zhan, and Hui Zhang. A clean slate 4d approach to network control and management. *SIGCOMM Comput. Commun. Rev.*, 35(5):41–54, Oct 2005.

[41] Martin Casado, Tal Garfinkel, Aditya Akella, Michael J. Freedman, Dan Boneh, Nick McKeown, and Scott Shenker. Sane: a protection architecture for enterprise networks. In *Proceedings of the 15th conference on USENIX Security Symposium - Volume 15*, USENIX-SS'06, Berkeley, CA, USA, 2006. USENIX Association.

[42] Martin Casado, Michael J. Freedman, Justin Pettit, Jianying Luo, Nick Mckeown, and Scott Shenker. Ethane: Taking control of the enterprise. In *In SIGCOMM Computer Comm. Rev*, New York, NY, USA, 2007. ACM.

[43] Changhoon Kim, Matthew Caesar, and Jennifer Rexford. Floodless in seattle: a scalable ethernet architecture for large enterprises. In *SIGCOMM '08: Proceedings of the ACM SIGCOMM 2008 conference on Data communication*, pages 3–14, New York, NY, USA, 2008. ACM.

[44] Greg Goth. Software-defined networking could shake up more than packets. *IEEE Internet Computing*, pages 6–9, 2011.

[45] ngel Leonardo Valdivieso Caraguay, Alberto Benito Peral, Lorena Isabel Barona Lpez, and Luis Javier Garca Villalba. Sdn: Evolution and opportunities in the development iot applications. *International Journal of Distributed Sensor Networks*, 2014, 2014.

[46] IBM. Ibm software defined networking in the new business frontier, Jul 2015.

[47] Joseph Packy Laverty, David Wood, and John Turchek. Software defined networking (sdn) network virtualization for the is curriculum? In *2014 Proceedings of the Information Systems Educators Conference*, volume 31. EDSIG (Education Special Interest Group of the AITP) and FITE (Foundation for Information Technology Education), 2014.

[48] Aryan TaheriMonfared and Chunming Rong. Multi-tenant network monitoring based on software defined networking. In Robert Meersman, Herv Panetto, Tharam Dillon, Johann Eder, Zohra Bellahsene, Norbert Ritter, Pieter De Leenheer, and Deijing Dou, editors, *On the Move to Meaningful Internet Systems: OTM 2013 Conferences*, volume 8185 of *Lecture Notes in Computer Science*, pages 327–341. Springer Berlin Heidelberg, 2013.

[49] J. Wickboldt, W. De Jesus, P. Isolani, C. Both, J. Rochol, and L. Granville. Software-defined networking: management requirements and challenges. *Communications Magazine, IEEE*, 53(1):278–285, Jan 2015.

[50] V.K. Gurbani, M. Scharf, T.V. Lakshman, V. Hilt, and E. Marocco. Abstracting network state in software defined networks (sdn) for rendezvous services. In *Communications (ICC), 2012 IEEE International Conference on*, pages 6627–6632, Jun 2012.

[51] Rob Sherwood, Glen Gibb, Kok-kiong Yap, Guido Appenzeller, Martin Casado, Nick Mckeown, and Guru Parulkar. Flowvisor: A network virtualization layer. Technical Report Openflow-tr-2009-1 Openflow-tr-2009-1, Stanford University, Stanford, CA, 2009.

[52] Jeffrey C. Mogul and Paul Congdon. Hey, you darned counters!: get off my asic! In *Proceedings of the first workshop on Hot topics in software defined networks*, HotSDN '12, pages 25–30, New York, NY, USA, 2012. ACM.

[53] Open Networking Foundation (ONF). Sdn architecture overview. `https://www.opennetworking.org`, Dec 2013. version 1.0.

[54] Brandon Heller, Colin Scott, Nick McKeown, Scott Shenker, Andreas Wundsam, Hongyi Zeng, Sam Whitlock, Vimalkumar Jeyakumar, Nikhil Handigol, James McCauley, Kyriakos Zarifis, and Peyman Kazemian. Leveraging sdn layering to systematically troubleshoot networks. In *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, HotSDN '13, pages 37–42, New York, NY, USA, 2013. ACM.

[55] Natasha Gude, Teemu Koponen, Justin Pettit, Ben Pfaff, Martín Casado, Nick McKeown, and Scott Shenker. Nox: towards an operating system for networks. *SIGCOMM Comput. Commun. Rev.*, 38(3):105–110, Jul 2008.

[56] Saro Velrajan. Application-aware routing in software-defined networking. `https://www.aricent.com`, 2013.

[57] Hesham Mekky, Fang Hao, Sarit Mukherjee, Zhi-Li Zhang, and T.V. Lakshman. Application-aware data plane processing in sdn. In *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking*, HotSDN '14, pages 13–18, New York, NY, USA, 2014. ACM.

[58] MRV Communications. Application-aware networking at a glance. `http://www.mrv.com`, 2013.

[59] Zafar Ayyub Qazi, Jeongkeun Lee, Tao Jin, Gowtham Bellala, Manfred Arndt, and Guevara Noubir. Application-awareness in sdn. In *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, SIGCOMM '13, pages 487–488, New York, NY, USA, 2013. ACM.

[60] K. Phemius, M. Bouet, and J. Leguay. Disco: Distributed multi-domain sdn controllers. In *Network Operations and Management Symposium (NOMS), 2014 IEEE*, pages 1–4, May 2014.

[61] H. Yin, H. Xie, D. Lopez, P. Aranda, and R. Sidi. Sdni: A message exchange protocol for software defined networks (sdns) across multiple domains. Internet-Draft draft-yin-sdn-sdni-00, Internet Engineering Task Force, Dec 2012. Work in progress.

[62] Hyojoon Kim and N. Feamster. Improving network management with software defined networking. *Communications Magazine, IEEE*, 51(2):114–119, Feb 2013.

[63] Aurojit Panda, Colin Scott, Ali Ghodsi, Teemu Koponen, and Scott Shenker. Cap for networks. In *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, HotSDN '13, pages 91–96, New York, NY, USA, 2013. ACM.

[64] Ian F. Akyildiz, Ahyoung Lee, Pu Wang, Min Luo, and Wu Chou. A roadmap for traffic engineering in sdn-openflow networks. *Comput. Netw.*, 71:1–30, Oct 2014.

[65] Yohei Kuga, Takeshi Matsuya, Hiroaki Hazeyama, Kenjiro Cho, and Osamu Nakamura. Etherpipe: An ethernet character device for network scripting. In *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, HotSDN '13, pages 61–66, New York, NY, USA, 2013. ACM.

[66] A. Doria, F. Hellstrand, K. Sundell, and T. Worster. General switch management protocol (gsmp). RFC 3292, Jun 2002.

[67] H. Khosravi and T. Anderson. Requirements for separation of ip control and forwarding. RFC 3654 (Informational), Nov 2003.

[68] Barath Raghavan, Martín Casado, Teemu Koponen, Sylvia Ratnasamy, Ali Ghodsi, and Scott Shenker. Software-defined internet architecture: decoupling architecture from infrastructure. In *Proceedings of the 11th ACM Workshop on Hot Topics in Networks*, pages 43–48, New York, NY, USA, 2012. ACM.

[69] Muhammad Shahbaz and Nick Feamster. The case for an intermediate representation for programmable data planes. In *Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research*, SOSR '15, pages 3:1–3:6, New York, NY, USA, 2015. ACM.

[70] A. Lara, A. Kolasani, and B. Ramamurthy. Network innovation using openflow: A survey. *Communications Surveys Tutorials, IEEE*, 16(1):493–512, Jan 2014.

[71] Vasileios Kotronis, Xenofontas Dimitropoulos, and Bernhard Ager. Outsourcing the routing control logic: better internet routing based on sdn principles. In *Proceedings of the 11th ACM Workshop on Hot Topics in Networks*, pages 55–60, New York, NY, USA, 2012. ACM.

[72] Glen Gibb, Hongyi Zeng, and Nick McKeown. Outsourcing network functionality. In *Proceedings of the first workshop on Hot topics in software defined networks*, HotSDN '12, New York, NY, USA, 2012. ACM.

[73] Mohammad Al-Fares, Alexander Loukissas, and Amin Vahdat. A scalable, commodity data center network architecture. In *Proceedings of the ACM SIGCOMM 2008 conference on Data communication*, SIGCOMM '08, pages 63–74, New York, NY, USA, 2008. ACM.

[74] Andreas Wundsam, Dan Levin, Srini Seetharaman, and Anja Feldmann. Ofrewind: enabling record and replay troubleshooting for networks. In *Proceedings of the 2011 USENIX conference on USENIX annual technical conference*, USENIXATC'11, pages 29–29, Berkeley, CA, USA, 2011. USENIX Association.

[75] Kanak Agarwal, Eric Rozner, Colin Dixon, and John Carter. Sdn traceroute: Tracing sdn forwarding without changing network behavior. In *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking*, HotSDN '14, pages 145–150, New York, NY, USA, 2014. ACM.

[76] Arne Schwabe and Holger Karl. Using mac addresses as efficient routing labels in data centers. In *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking*, HotSDN '14, pages 115–120, New York, NY, USA, 2014. ACM.

[77] Albert G. Greenberg, James R. Hamilton, Navendu Jain, Srikanth Kandula, Changhoon Kim, Parantap Lahiri, David A. Maltz, Parveen Patel, and Sudipta Sengupt. Vl2: A scalable and flexible data center network. In *ACM SIGCOMM Conference - SIGCOMM 2009*, pages 51–62, New York, NY, USA, 2009. ACM.

[78] Chen Chen, Changbin Liu, Pingkai Liu, Boon Thau Loo, and Ling Ding. A scalable multi-datacenter layer-2 network architecture. In *Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research*, SOSR '15, pages 8:1–8:12, New York, NY, USA, 2015. ACM.

[79] Sangeetha Abdu Jyothi, Mo Dong, and P. Brighten Godfrey. Towards a flexible data center fabric with source routing. In *Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research*, SOSR '15, pages 10:1–10:8, New York, NY, USA, 2015. ACM.

[80] Luyuan Fang, Fabio Chiussi, Deepak Bansal, Vijay Gill, Tony Lin, Jeff Cox, and Gary Ratterree. Hierarchical sdn for the hyper-scale, hyper-elastic data center and cloud. In *Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research*, SOSR '15, pages 7:1–7:13, New York, NY, USA, 2015. ACM.

[81] Zhongjin Liu, Yong Li, Li Su, Depeng Jin, and Lieguang Zeng. M2cloud: Software defined multi-site data center network control framework for multi-tenant. In *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, SIGCOMM '13, pages 517–518, New York, NY, USA, 2013. ACM.

[82] Soheil Hassas Yeganeh and Yashar Ganjali. Kandoo: a framework for efficient and scalable offloading of control applications. In *Proceedings of the first workshop on Hot topics in software defined networks*, HotSDN '12, pages 19–24, New York, NY, USA, 2012. ACM.

[83] Chi-Yao Hong, Srikanth Kandula, Ratul Mahajan, Ming Zhang, Vijay Gill, Mohan Nanduri, and Roger Wattenhofer. Achieving high utilization with software-driven wan. In *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, SIGCOMM '13, pages 15–26, New York, NY, USA, 2013. ACM.

[84] Li Erran Li, Z. Morley Mao, and Jennifer Rexford. Toward software-defined cellular networks. In *Proc. European Workshop on Software Defined Networking*, Oct 2012.

[85] M. Bouet, J. Leguay, and V. Conan. Cost-based placement of virtualized deep packet inspection functions in sdn. In *Military Communications Conference, MILCOM 2013 - 2013 IEEE*, pages 992–997, Nov 2013.

[86] Anat Bremler-Barr, Yotam Harchol, David Hay, and Yaron Koral. Deep packet inspection as a service. In *Proceedings of the 10th ACM International on Conference on Emerging Networking Experiments and Technologies*, CoNEXT '14, pages 271–282, New York, NY, USA, 2014. ACM.

[87] Lalith Suresh, Julius Schulz-Zander, Ruben Merz, Anja Feldmann, and Teresa Vazao. Towards programmable enterprise wlans with odin. In *Proceedings of the first workshop on Hot topics in software defined networks*, HotSDN '12, pages 115–120, New York, NY, USA, 2012. ACM.

[88] Nachikethas A. Jagadeesan and Bhaskar Krishnamachari. Software-defined networking paradigms in wireless networks: A survey. *ACM Comput. Surv.*, 47(2):27:1–27:11, Nov 2014.

[89] Julius Schulz-Zander, Carlos Mayer, Bogdan Ciobotaru, Stefan Schmid, and Anja Feldmann. Opensdwn: Programmatic control over home and enterprise wifi. In *Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research*, SOSR '15, pages 16:1–16:12, New York, NY, USA, 2015. ACM.

[90] Manu Bansal, Jeffrey Mehlman, Sachin Katti, and Philip Levis. Openradio: a programmable wireless dataplane. In *Proceedings of the first workshop on Hot topics in software defined networks*, HotSDN '12, pages 109–114, New York, NY, USA, 2012.

[91] Aditya Gudipati, Daniel Perry, Li Erran Li, and Sachin Katti. Softran: Software defined radio access network. In *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, HotSDN '13, pages 25–30, New York, NY, USA, 2013. ACM.

[92] Mao Yang, Yong Li, Depeng Jin, Li Su, Shaowu Ma, and Lieguang Zeng. Open-ran: A software-defined ran architecture via virtualization. In *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, SIGCOMM '13, pages 549–550, New York, NY, USA, 2013. ACM.

[93] Kanthi Nagaraj and Sachin Katti. Procel: Smart traffic handling for a scalable software epc. In *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking*, HotSDN '14, pages 43–48, New York, NY, USA, 2014. ACM.

[94] M. Kind, F. Westphal, A. Gladisch, and S. Topp. Splitarchitecture: Applying the software defined networking concept to carrier networks. In *World Telecommunications Congress (WTC), 2012*, pages 1 –6, Mar 2012.

[95] D. Staessens, S. Sharma, D. Colle, M. Pickavet, and P. Demeester. Software defined networking: Meeting carrier grade requirements. In *Local Metropolitan Area Networks (LANMAN), 2011 18th IEEE Workshop on*, pages 1–6, Oct 2011.

[96] Yu Hua, Xue Liu, and Dan Feng. Smart in-network deduplication for storage-aware sdn. In *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, SIGCOMM '13, pages 509–510, New York, NY, USA, 2013. ACM.

[97] Dennis M. Volpano, Xin Sun, and Geoffrey G. Xie. Towards systematic detection and resolution of network control conflicts. In *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking*, HotSDN '14, pages 67–72, New York, NY, USA, 2014. ACM.

[98] Marshini Chetty and Nick Feamster. Refactoring network infrastructure to improve manageability: a case study of home networking. *SIGCOMM Comput. Commun. Rev.*, 42(3):54–61, Jun 2012.

[99] Yiannis Yiakoumis, Kok-Kiong Yap, Sachin Katti, Guru Parulkar, and Nick McKeown. Slicing home networks. In *Proceedings of the 2nd ACM SIGCOMM workshop on Home networks*, HomeNets '11, pages 1–6, New York, NY, USA, 2011. ACM.

[100] National. Global environment for network innovations (geni). Available from: http://www.geni.net/, 2006.

[101] Minseok Lee, Younggi Kim, and Younghee Lee. A home cloud-based home network auto-configuration using sdn. In *Networking, Sensing and Control (ICNSC), 2015 IEEE 12th International Conference on*, pages 444–449, Apr 2015.

[102] Kuang-Ching Wang. Floodlight. `http://www.projectfloodlight.org/floodlight`, Apr 2013.

[103] David Erickson. The beacon openflow controller. In *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, HotSDN '13, pages 13–18, New York, NY, USA, 2013. ACM.

[104] Nick Shelly, Ethan J. Jackson, Teemu Koponen, Nick McKeown, and Jarno Rajahalme. Flow caching for high entropy packet fields. In *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking*, HotSDN '14, pages 151–156, New York, NY, USA, 2014. ACM.

[105] Srinivas Narayana, Jennifer Rexford, and David Walker. Compiling path queries in software-defined networks. In *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking*, HotSDN '14, pages 181–186, New York, NY, USA, 2014. ACM.

[106] Ye Yu, Chen Qian, and Xin Li. Distributed and collaborative traffic monitoring in software defined networks. In *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking*, HotSDN '14, pages 85–90, New York, NY, USA, 2014. ACM.

[107] B. Martini, F. Paganelli, A.A. Mohammed, M. Gharbaoui, A. Sgambelluri, and P. Castoldi. Sdn controller for context-aware data delivery in dynamic service chaining. In *Network Softwarization (NetSoft), 2015 1st IEEE Conference on*, pages 1–5, Apr 2015.

[108] Christian Esteve Rothenberg, Marcelo Ribeiro Nascimento, Marcos Rogerio Salvador, Carlos Nilton Araujo Corrêa, Sidney Cunha de Lucena, and Robert Raszuk. Revisiting routing control platforms with the eyes and muscles of software-defined networking. In *Proceedings of the first workshop on Hot topics in software defined networks*, HotSDN '12, pages 13–18, New York, NY, USA, 2012. ACM.

[109] Heng Pan, Hongtao Guan, Junjie Liu, Wanfu Ding, Chengyong Lin, and Gaogang Xie. The flowadapter: Enable flexible multi-table processing on legacy hardware. In *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, HotSDN '13, pages 85–90, New York, NY, USA, 2013. ACM.

[110] Zhou Jingjing, Cheng Di, Wang Weiming, Jin Rong, and Wu Xiaochun. The deployment of routing protocols in distributed control plane of sdn. *The Scientific World Journal*, 2014, 2014.

[111] N. van Adrichem, B. van Asten, and F. A. Kuipers. Fast recovery in software-defined networks. In *European Workshop on Software Defined Networking (EWSDN 2014)*, Budapest (Hungary), Sep 2014.

[112] Maciej Kuźniar, Peter Perešíni, Nedeljko Vasić, Marco Canini, and Dejan Kostić. Automatic failure recovery for software-defined networks. In *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, HotSDN '13, pages 159–160, New York, NY, USA, 2013. ACM.

[113] Dan Williams and Hani Jamjoom. Cementing high availability in openflow with rulebricks. In *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, HotSDN '13, pages 139–144, New York, NY, USA, 2013. ACM.

[114] Sachin Sharma, Dimitri Staessens, Didier Colle, Mario Pickavet, and Piet Demeester. Automatic configuration of routing control platforms in openflow networks. In *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, SIGCOMM '13, pages 491–492, New York, NY, USA, 2013. ACM.

[115] Matt Davy. A case for expanding openflow/sdn deployments on university campuses. `http://www.openflow.org`, 2011.

[116] Nick McKeown. Clean slate: An interdisciplinary research program. `http://cleanslate.stanford.edu`, 2015.

[117] Ben Pfaff, Justin Pettit, Teemu Koponen, Keith Amidon, Martin Casado, and Scott Shenker. Extending networking into the virtualization layer. In *8th ACM Workshop on Hot Topics inNetworks (HotNets-VIII)*, Oct 2009.

[118] Justin Pettit, Jesse Gross, Ben Pfaff, Martin Casado, and Simon Crosby. Virtual switching in an era of advanced edges. In *2nd Workshop on Data Center - Converged and Virtual Ethernet Switching (DC CAVES)*, Wrzburg, Germany, Sep 2010. DC CAVES.

[119] Abhinava Sadasivarao, Sharfuddin Syed, Ping Pan, Chris Liou, Andrew Lake, Chin Guok, and Inder Monga. Open transport switch: A software defined networking architecture for transport networks. In *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, HotSDN '13, pages 115–120, New York, NY, USA, 2013. ACM.

[120] M. Channegowda, R. Nejabati, and D. Simeonidou. Software-defined optical networks technology and infrastructure: Enabling software-defined optical network operations [invited]. *Optical Communications and Networking, IEEE/OSA Journal of*, 5(10):A274–A282, Oct 2013.

[121] Casimer DeCusatis. Reference architecture for multi-layer software defined optical data center networks. *Electronics*, 4(3):633, 2015.

[122] N. Kitsuwan, S. McGettrick, F. Slyne, D.B. Payne, and M. Ruffini. Independent transient plane design for protection in openflow-based networks. *Optical Communications and Networking, IEEE/OSA Journal of*, 7(4):264–275, Apr 2015.

[123] D. Simeonidou, R. Nejabati, and S. Azodolmolky. Enabling the future optical internet with openflow: A paradigm shift in providing intelligent optical network services. In *Transparent Optical Networks (ICTON), 2011 13th International Conference on*, pages 1–4, Jun 2011.

[124] Hui Yang, Yadi Cui, and Jie Zhang. Unified multi-layer among software defined multi-domain optical networks (invited). *Electronics*, 4(2):329, 2015.

[125] S.J.B. Yoo. Multi-domain cognitive optical software defined networks with market-driven brokers. In *Optical Communication (ECOC), 2014 European Conference on*, pages 1–3, Sep 2014.

[126] S. Tariq and M. Bassiouni. Qamo-sdn: Qos aware multipath tcp for software defined optical networks. In *Consumer Communications and Networking Conference (CCNC), 2015 12th Annual IEEE*, pages 485–491, Jan 2015.

[127] NetFPGA Team. Netfpga website, Mar 2012.

[128] Jad Naous, Glen Gibb, Sara Bolouki, and Nick McKeown. Netfpga: reusable router architecture for experimental research. In *Proceedings of the ACM workshop on Programmable routers for extensible services of tomorrow*, PRESTO '08, pages 1–7, New York, NY, USA, 2008. ACM.

[129] Jad Naous, David Erickson, G. Adam Covington, Guido Appenzeller, and Nick McKeown. Implementing an openflow switch on the netfpga platform. In *Proceedings of the 4th ACM/IEEE Symposium on Architectures for Networking and Communications Systems*, ANCS '08, pages 1–9, New York, NY, USA, 2008. ACM.

[130] Muhammad Bilal Anwer, Murtaza Motiwala, Mukarram bin Tariq, and Nick Feamster. Switchblade: a platform for rapid deployment of network protocols on programmable hardware. *SIGCOMM Comput. Commun. Rev.*, 40(4):183–194, Aug 2010.

[131] Robert Morris, Eddie Kohler, John Jannotti, and M. Frans Kaashoek. The click modular router. In *Proceedings of the seventeenth ACM symposium on Operating systems principles*, SOSP '99, pages 217–231, New York, NY, USA, 1999. ACM.

[132] Minlan Yu, Jennifer Rexford, Michael J. Freedman, and Jia Wang. Scalable flow-based networking with difane. *SIGCOMM Comput. Commun. Rev.*, 41(4), Aug 2010.

[133] Ying Zhang, Sriram Natarajan, Xin Huang, Neda Beheshti, and Ravi Manghirmalani. A compressive method for maintaining forwarding states in sdn controller. In *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking*, HotSDN '14, pages 139–144, New York, NY, USA, 2014. ACM.

[134] Andrew R. Curtis, Jeffrey C. Mogul, Jean Tourrilhes, Praveen Yalagandula, Puneet Sharma, and Sujata Banerjee. Devoflow: scaling flow management for high-performance networks. *SIGCOMM Comput. Commun. Rev.*, 41(4):254–265, Aug 2011.

[135] Martin Casado, Teemu Koponen, Daekyeong Moon, and Scott Shenker. Rethinking packet forwarding hardware. In *In Proceedings of ACM HotNets*, New York, NY, USA, 2008. ACM.

[136] Vyas Sekar, Norbert Egi, Sylvia Ratnasamy, Michael Reiter, and Guangyu Shi. Design and implementation of a consolidated middlebox architecture. Technical Report UCB/EECS-2011-110, EECS Department, University of California, Berkeley, Oct 2011.

[137] Aaron Gember, Prathmesh Prabhu, Zainab Ghadiyali, and Aditya Akella. Toward software-defined middlebox networking. In *Proceedings of the 11th ACM Workshop on Hot Topics in Networks*, HotNets-XI, pages 7–12, New York, NY, USA, 2012. ACM.

[138] Joao Martins, Mohamed Ahmed, Costin Raiciu, and Felipe Huici. Enabling fast, dynamic network processing with clickos. In *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, HotSDN '13, pages 67–72, New York, NY, USA, 2013. ACM.

[139] Bilal Anwer, Theophilus Benson, Nick Feamster, and Dave Levin. Programming slick network functions. In *Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research*, SOSR '15, pages 14:1–14:13, New York, NY, USA, 2015. ACM.

[140] Zafar Ayyub Qazi, Cheng-Chun Tu, Luis Chiang, Rui Miao, Vyas Sekar, and Minlan Yu. Simple-fying middlebox policy enforcement using sdn. *SIGCOMM Comput. Commun. Rev.*, 43(4):27–38, Aug 2013.

[141] Aaron Gember, Robert Grandl, Junaid Khalid, and Aditya Akella. Design and implementation of a framework for software-defined middlebox networking. *SIGCOMM Comput. Commun. Rev.*, 43(4):467–468, Aug 2013.

[142] Giuseppe Bianchi, Marco Bonola, Antonio Capone, and Carmelo Cascone. Openstate: programming platform-independent stateful openflow applications inside the switch. *Computer Communication Review*, 44(2):44–51, 2014.

[143] Masoud Moshref, Apoorv Bhargava, Adhip Gupta, Minlan Yu, and Ramesh Govindan. Flow-level state transition as a new switch primitive for sdn. In *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking*, HotSDN '14, pages 61–66, New York, NY, USA, 2014. ACM.

[144] Minlan Yu, Andreas Wundsam, and Muruganantham Raju. Nosix: A lightweight portability layer for the sdn os. *Computer Communication Review*, 44(2):28–35, 2014.

[145] Hani Jamjoom, Dan Williams, and Upendra Sharma. Don't call them middleboxes, call them middlepipes. In *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking*, HotSDN '14, pages 19–24, New York, NY, USA, 2014. ACM.

[146] Angela Chiu, Vijay Gopalakrishnan, Bo Han, Murad Kablan, Oliver Spatscheck, Chengwei Wang, and Yang Xu. Edgeplex: Decomposing the provider edge for flexibilty and reliability. In *Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research*, SOSR '15, pages 15:1–15:6, New York, NY, USA, 2015. ACM.

[147] Michio Honda, Felipe Huici, Giuseppe Lettieri, and Luigi Rizzo. mswitch: A highly-scalable, modular software switch. In *Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research*, SOSR '15, pages 1:1–1:13, New York, NY, USA, 2015. ACM.

[148] Teemu Koponen, Martin Casado, Natasha Gude, Jeremy Stribling, Leon Poutievski, Min Zhu, Rajiv Ramanathan, Yuichiro Iwata, Hiroaki Inoue, Takayuki Hama, and Scott Shenker. Onix: a distributed control platform for large-scale production networks. In *Proceedings of the 9th USENIX conference on Operating systems design and implementation*, OSDI'10, pages 1–6, Berkeley, CA, USA, 2010. USENIX Association.

[149] Dan Levin, Andreas Wundsam, Brandon Heller, Nikhil Handigol, and Anja Feldmann. Logically centralized?: state distribution trade-offs in software defined networks. In *Proceedings of the first workshop on Hot topics in software defined networks*, HotSDN '12, pages 1–6, New York, NY, USA, 2012. ACM.

[150] Diego Kreutz, Fernando M.V. Ramos, and Paulo Verissimo. Towards secure and dependable software-defined networks. In *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, HotSDN '13, pages 55–60, New York, NY, USA, 2013. ACM.

[151] Amin Tootoonchian and Yashar Ganjali. Hyperflow: a distributed control plane for openflow. In *Proceedings of the 2010 internet network management conference on Research on enterprise networking*, INM/WREN'10, pages 3–3, Berkeley, CA, USA, 2010. USENIX Association.

[152] Brandon Heller, Rob Sherwood, and Nick McKeown. The controller placement problem. In *Proceedings of the first workshop on Hot topics in software defined networks*, HotSDN '12, pages 7–12, New York, NY, USA, 2012. ACM.

[153] Francisco Javier Ros and Pedro Miguel Ruiz. Five nines of southbound reliability in software-defined networks. In *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking*, HotSDN '14, pages 31–36, New York, NY, USA, 2014. ACM.

[154] S. Lange, S. Gebert, J. Spoerhase, P. Rygielski, T. Zinner, S. Kounev, and Phuoc Tran-Gia. Specialized heuristics for the controller placement problem in large scale sdn networks. In *Teletraffic Congress (ITC 27), 2015 27th International*, pages 210–218, Sep 2015.

[155] S. Lange, S. Gebert, T. Zinner, P. Tran-Gia, D. Hock, M. Jarschel, and M. Hoffmann. Heuristic approaches to the controller placement problem in large scale sdn networks. *Network and Service Management, IEEE Transactions on*, 12(1):4–17, Mar 2015.

[156] Long Yao, Peilin Hong, Wen Zhang, Jianfei Li, and Dan Ni. Controller placement and flow based dynamic management problem towards sdn. In *Communication Workshop (ICCW), 2015 IEEE International Conference on*, pages 363–368, Jun 2015.

[157] Shan Gao, Sho Shimizu, Satoru Okamoto, and Naoaki Yamanaka. A high-speed routing engine for software defined network. *Journal of Selected Areas in Telecommunications (JSAT)*, pages 1–7, Aug 2012.

[158] Zheng Cai, Alan L. Cox, and T. S. Eugene Ng. Maestro: A system for scalable openflow control. Rice University Technical Report TR10-08, Rice University, Houston, TX, 2010.

[159] Charalampos Rotsos, Nadi Sarrar, Steve Uhlig, Rob Sherwood, and Andrew W. Moore. Oflops: an open framework for openflow switch evaluation. In *Proceedings of the 13th international conference on Passive and Active Measurement*, PAM'12, pages 85–95, Berlin, Heidelberg, 2012. Springer-Verlag.

[160] Danny Yuxing Huang, Kenneth Yocum, and Alex C. Snoeren. High-fidelity switch models for software-defined network emulation. In *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, HotSDN '13, pages 43–48, New York, NY, USA, 2013. ACM.

[161] Maciej Kuzniar, Peter Peresini, Marco Canini, Daniele Venzano, and Dejan Kostic. A soft way for openflow switch interoperability testing. In *Proceedings of the 8th international conference on Emerging networking experiments and technologies*, CoNEXT '12, pages 265–276, New York, NY, USA, 2012. ACM.

[162] Keqiang He, Junaid Khalid, Aaron Gember-Jacobson, Sourav Das, Chaithan Prakash, Aditya Akella, Li Erran Li, and Marina Thottan. Measuring control plane latency in sdn-enabled switches. In *Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research*, SOSR '15, pages 25:1–25:6, New York, NY, USA, 2015. ACM.

[163] Amin Tootoonchian, Sergey Gorbunov, Yashar Ganjali, Martin Casado, and Rob Sherwood. On controller performance in software-defined networks. In *Proceedings of the 2nd USENIX conference on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services*, Hot-ICE'12, pages 10–10, Berkeley, CA, USA, 2012. USENIX Association.

[164] Stephen Gutz, Alec Story, Cole Schlesinger, and Nate Foster. Splendid isolation: a slice abstraction for software-defined networks. In *Proceedings of the first workshop on Hot topics in software defined networks*, HotSDN '12, pages 79–84, New York, NY, USA, 2012. ACM.

[165] Laurent Vanbever, Joshua Reich, Theophilus Benson, Nate Foster, and Jennifer Rexford. Hotswap: Correct and efficient controller upgrades for software-defined networks. In *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, HotSDN '13, pages 133–138, New York, NY, USA, 2013. ACM.

[166] Ilya Baldin, Shu Huang, and Rajesh Gopidi. A resource delegation framework for software defined networks. In *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking*, HotSDN '14, pages 49–54, New York, NY, USA, 2014. ACM.

[167] Ali Al-Shabibi, Marc De Leenheer, Matteo Gerola, Ayaka Koshibe, Guru Parulkar, Elio Salvadori, and Bill Snow. Openvirtex: Make your virtual sdns programmable. In *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking*, HotSDN '14, pages 25–30, New York, NY, USA, 2014. ACM.

[168] A. Blenk, A. Basta, and W. Kellerer. Hyperflex: An sdn virtualization architecture with flexible hypervisor function allocation. In *Integrated Network Management (IM), 2015 IFIP/IEEE International Symposium on*, pages 397–405, May 2015.

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

[170] Dmitry Drutskoy, Eric Keller, and Jennifer Rexford. Scalable network virtualization in software-defined networks. *Internet Computing, IEEE*, 17(2):20–27, Mar 2013.

[171] Soudeh Ghorbani and Brighten Godfrey. Towards correct network virtualization. In *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking*, HotSDN '14, pages 109–114, New York, NY, USA, 2014. ACM.

[172] Sergey Guenender, Katherine Barabash, Yaniv Ben-Itzhak, Anna Levin, Eran Raichstein, and Liran Schour. Noencap: Overlay network virtualization with no encapsulation overheads. In *Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research*, SOSR '15, pages 9:1–9:7, New York, NY, USA, 2015. ACM.

[173] Fang Hao, T. V. Lakshman, Sarit Mukherjee, and Haoyu Song. Enhancing dynamic cloud-based services using network virtualization. *SIGCOMM Comput. Commun. Rev.*, 40(1):67–74, Jan 2010.

[174] Peyman Kazemian, George Varghese, and Nick McKeown. Header space analysis: static checking for networks. In *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*, NSDI'12, pages 9–9, Berkeley, CA, USA, 2012. USENIX Association.

[175] Stephanos Matsumoto, Samuel Hitz, and Adrian Perrig. Fleet: Defending sdns from malicious administrators. In *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking*, HotSDN '14, pages 103–108, New York, NY, USA, 2014. ACM.

[176] Abhishek Dwaraki, Srini Seetharaman, Sriram Natarajan, and Tilman Wolf. Gitflow: Flow revision management for software-defined networks. In *Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research*, SOSR '15, pages 6:1–6:6, New York, NY, USA, 2015. ACM.

[177] Abhishek Chanda, Cedric Westphal, and Dipankar Raychaudhuri. Content based traffic engineering in software defined information centric networks. `http://arxiv.org/abs/1301.7517`, 2013.

[178] S. Agarwal, M. Kodialam, and T. V. Lakshman. Traffic engineering in software defined networks. In *INFOCOM, 2013 Proceedings IEEE*, pages 2211–2219. IEEE, Apr 2013.

[179] Eric Keller, Soudeh Ghorbani, Matt Caesar, and Jennifer Rexford. Live migration of an entire network (and its hosts). In *Proceedings of the 11th ACM Workshop on Hot Topics in Networks*, HotNets-XI, pages 109–114, New York, NY, USA, 2012. ACM.

[180] Mohammad Al-Fares, Sivasankar Radhakrishnan, Barath Raghavan, Nelson Huang, and Amin Vahdat. Hedera: dynamic flow scheduling for data center networks. In *Proceedings of the 7th USENIX conference on Networked systems design and implementation*, NSDI'10, pages 19–19, Berkeley, CA, USA, 2010. USENIX Association.

[181] Kang Xi, Yulei Liu, and H.J. Chao. Enabling flow-based routing control in data center networks using probe and ecmp. In *Computer Communications Workshops (INFOCOM WKSHPS), 2011 IEEE Conference on*, pages 608–613, 2011.

[182] Nikhil Handigol, Srinivasan Seetharaman, Nick Mckeown, and Ramesh Johari. Plug-n-serve: Load-balancing web traffic using openflow, 2008.

[183] S. Das, A.R. Sharafat, G. Parulkar, and N. McKeown. Mpls with a simple open control plane. In *Optical Fiber Communication Conference and Exposition (OFC/NFOEC) and the National Fiber Optic Engineers Conference*, pages 1 –3, Washington, DC, USA, Mar 2011. IEEE.

[184] Ali Reza Sharafat, Saurav Das, Guru Parulkar, and Nick McKeown. Mpls-te and mpls vpns with openflow. *SIGCOMM Comput. Commun. Rev.*, 41(4):452–453, Aug 2011.

[185] Monia Ghobadi, Soheil Hassas Yeganeh, and Yashar Ganjali. Rethinking end-to-end congestion control in software-defined networks. In *Proceedings of the 11th ACM Workshop on Hot Topics in Networks*, pages 61–66, New York, NY, USA, 2012. ACM.

[186] Bo Yan, Yang Xu, Hongya Xing, Kang Xi, and H. Jonathan Chao. Cab: A reactive wildcard rule caching system for software-defined networks. In *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking*, HotSDN '14, pages 163–168, New York, NY, USA, 2014. ACM.

[187] Richard Wang, Dana Butnariu, and Jennifer Rexford. Openflow-based server load balancing gone wild, 2010.

[188] A. Craig, B. Nandy, I. Lambadaris, and P. Ashwood-Smith. Load balancing for multicast traffic in sdn using real-time link cost modification. In *Communications (ICC), 2015 IEEE International Conference on*, pages 5789–5795, Jun 2015.

[189] Gergely Pongrácz, László Molnár, Zoltán Lajos Kis, and Zoltán Turányi. Cheap silicon: A myth or reality? picking the right data plane hardware for software defined networking. In *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, HotSDN '13, pages 103–108, New York, NY, USA, 2013. ACM.

[190] Peng Sun, Laurent Vanbever, and Jennifer Rexford. Scalable programmable inbound traffic engineering. In *Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research*, SOSR '15, pages 12:1–12:7, New York, NY, USA, 2015. ACM.

[191] Seugwon Shin, Phillip Porras, Vinod Yegneswaran, Martin Fong, Guofei Gu, and Mabry Tyson. Fresco: Modular composable security services for software-defined networks. In *Proceedings of the ISOC Network and Distributed System Security Symposium*, ISOC, Geneva, Switzerland, 2013. Internet Society.

[192] Philip Porras, Seungwon Shin, Vinod Yegneswaran, Martin Fong, Mabry Tyson, and Guofei Gu. A security enforcement kernel for openflow networks. In *Proceedings of the first workshop on Hot topics in software defined networks*, HotSDN '12, pages 121–126, New York, NY, USA, 2012. ACM.

[193] Zhiyuan Hu, Mingwen Wang, Xueqiang Yan, Yueming Yin, and Zhigang Luo. A comprehensive security architecture for sdn. In *Intelligence in Next Generation Networks (ICIN), 2015 18th International Conference on*, pages 30–37, Feb 2015.

[194] Hongxin Hu, Wonkyu Han, Gail-Joon Ahn, and Ziming Zhao. Flowguard: Building robust firewalls for software-defined networks. In *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking*, HotSDN '14, pages 97–102, New York, NY, USA, 2014. ACM.

[195] Y. Ben-Itzhak, K. Barabash, R. Cohen, A. Levin, and E. Raichstein. Enforsdn: Network policies enforcement with sdn. In *Integrated Network Management (IM), 2015 IFIP/IEEE International Symposium on*, pages 80–88, May 2015.

[196] S. Civanlar, M. Parlakisik, A.M. Tekalp, B. Gorkemli, B. Kaytaz, and E. Onem. A qos-enabled openflow environment for scalable video streaming. In *GLOBECOM Workshops (GC Wkshps), 2010 IEEE*, pages 351–356. IEEE, Dec 2010.

[197] H.E. Egilmez, B. Gorkemli, A.M. Tekalp, and S. Civanlar. Scalable video streaming over openflow networks: An optimization framework for qos routing. In *Image Processing (ICIP), 2011 18th IEEE International Conference on*, pages 2241 –2244, Brussels, Belguim, Sep 2011. IEEE.

[198] S. Laga, T. Van Cleemput, F. Van Raemdonck, F. Vanhoutte, N. Bouten, M. Claeys, and F. De Turck. Optimizing scalable video delivery through openflow layer-based routing. In *Network Operations and Management Symposium (NOMS), 2014 IEEE*, pages 1–4, May 2014.

[199] K. A. Noghani and M. O. Sunay. Streaming multicast video over software-defined networks. In *Mobile Ad Hoc and Sensor Systems (MASS), 2014 IEEE 11th International Conference on*, pages 551–556, Oct 2014.

[200] H. Owens II and A. Durresi. Video over software-defined networking (vsdn). In *Proceedings of the 16th International Conference on Network-Based Information Systems (NBiS'2013)*, pages 44–51, Sep 2013.

[201] Thomas G. Edwards and Warren Belkin. Using sdn to facilitate precisely timed actions on real-time data streams. In *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking*, HotSDN '14, pages 55–60, New York, NY, USA, 2014. ACM.

[202] Andreas Voellmy and Paul Hudak. Nettle: taking the sting out of programming network routers. In *Proceedings of the 13th international conference on Practical aspects of declarative languages*, PADL'11, pages 235–249, Berlin, Heidelberg, 2011. Springer-Verlag.

[203] Nate Foster, Rob Harrison, Michael J. Freedman, Christopher Monsanto, Jennifer Rexford, Alec Story, and David Walker. Frenetic: a network programming language. *SIGPLAN Not.*, 46(9):279–291, Sep 2011.

[204] Anirudh Sivaraman, Changhoon Kim, Ramkumar Krishnamoorthy, Advait Dixit, and Mihai Budiu. Dc.p4: Programming the forwarding plane of a datacenter switch. In *Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research*, SOSR '15, pages 2:1–2:8, New York, NY, USA, 2015. ACM.

[205] Andreas Voellmy, Hyojoon Kim, and Nick Feamster. Procera: a language for high-level reactive network control. In *Proceedings of the first workshop on Hot topics in software defined networks*, HotSDN '12, pages 43–48, New York, NY, USA, 2012. ACM.

[206] Timothy Nelson, Arjun Guha, Daniel J. Dougherty, Kathi Fisler, and Shriram Krishnamurthi. A balance of power: expressive, analyzable controller programming. In *HotSDN*, pages 79–84. ACM, 2013.

[207] Tim Nelson, Andrew D. Ferguson, Da Yu, Rodrigo Fonseca, and Shriram Krishnamurthi. Exodus: Toward automatic migration of enterprise network configurations to sdns. In *Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research*, SOSR '15, pages 13:1–13:7, New York, NY, USA, 2015. ACM.

[208] Mark Reitblatt, Marco Canini, Arjun Guha, and Nate Foster. Fattire: Declarative fault tolerance for software-defined networks. In *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, HotSDN '13, pages 109–114, New York, NY, USA, 2013. ACM.

[209] Christopher Monsanto, Nate Foster, Rob Harrison, and David Walker. A compiler and run-time system for network programming languages. In *Proceedings of the 39th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, POPL '12, pages 217–230, New York, NY, USA, 2012. ACM.

[210] Shambwaditya Saha, Santhosh Prabhu, and P. Madhusudan. Netgen: Synthesizing data-plane configurations for network policies. In *Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research*, SOSR '15, pages 17:1–17:6, New York, NY, USA, 2015. ACM.

[211] Andreas Voellmy, Junchang Wang, Y Richard Yang, Bryan Ford, and Paul Hudak. Maple: Simplifying sdn programming using algorithmic policies. *SIGCOMM Comput. Commun. Rev.*, 43(4):87–98, Aug 2013.

[212] Bob Lantz, Brandon Heller, and Nick McKeown. A network in a laptop: rapid prototyping for software-defined networks. In *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, Hotnets-IX, pages 19:1–19:6, New York, NY, USA, 2010. ACM.

[213] Jiaqi Yan and Dong Jin. Vt-mininet: Virtual-time-enabled mininet for scalable and accurate software-define network emulation. In *Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research*, SOSR '15, pages 27:1–27:7, New York, NY, USA, 2015. ACM.

[214] Ahmed Khurshid, Wenxuan Zhou, Matthew Caesar, and P. Brighten Godfrey. Veriflow: verifying network-wide invariants in real time. In *Proceedings of the first workshop on Hot topics in software defined networks*, HotSDN '12, pages 49–54, New York, NY, USA, 2012. ACM.

[215] Ryan Beckett, Xuan Kelvin Zou, Shuyuan Zhang, Sharad Malik, Jennifer Rexford, and David Walker. An assertion language for debugging sdn applications. In *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking*, HotSDN '14, pages 91–96, New York, NY, USA, 2014. ACM.

[216] Naga Praveen Katta, Jennifer Rexford, and David Walker. Incremental consistent updates. In *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, HotSDN '13, pages 49–54, New York, NY, USA, 2013. ACM.

[217] Xitao Wen, Chunxiao Diao, Xun Zhao, Yan Chen, Li Erran Li, Bo Yang, and Kai Bu. Compiling minimum incremental update for modular sdn languages. In *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking*, HotSDN '14, pages 193–198, New York, NY, USA, 2014. ACM.

[218] Jeremie Miserez, Pavol Bielik, Ahmed El-Hassany, Laurent Vanbever, and Martin Vechev. Sdnracer: Detecting concurrency violations in software-defined networks. In *Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research*, SOSR '15, pages 22:1–22:7, New York, NY, USA, 2015. ACM.

[219] Marco Canini, Daniele Venzano, Peter Perešíni, Dejan Kostić, and Jennifer Rexford. A nice way to test openflow applications. In *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*, NSDI'12, pages 10–10, Berkeley, CA, USA, 2012. USENIX Association.

[220] Nikhil Handigol, Brandon Heller, Vimalkumar Jeyakumar, David Maziéres, and Nick McKeown. Where is the debugger for my software-defined network? In *Proceedings of the first workshop on Hot topics in software defined networks*, HotSDN '12, pages 55–60, New York, NY, USA, 2012. ACM.

[221] Hui Zhang, Cristian Lumezanu, Junghwan Rhee, Nipun Arora, Qiang Xu, and Guofei Jiang. Enabling layer 2 pathlet tracing through context encoding in software-defined networking. In *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking*, HotSDN '14, pages 169–174, New York, NY, USA, 2014. ACM.

[222] Mukta Gupta, Joel Sommers, and Paul Barford. Fast, accurate simulation for sdn prototyping. In *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, HotSDN '13, pages 31–36, New York, NY, USA, 2013. ACM.

[223] Seyed K. Fayaz and Vyas Sekar. Testing stateful and dynamic data planes with flowtest. In *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking*, HotSDN '14, pages 79–84, New York, NY, USA, 2014. ACM.

[224] Praveen Tammana, Rachit Agarwal, and Myungjin Lee. Cherrypick: Tracing packet trajectory in software-defined datacenter networks. In *Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research*, SOSR '15, pages 23:1–23:7, New York, NY, USA, 2015. ACM.

[225] István Pelle, Tamás Lévai, Felicián Németh, and András Gulyás. One tool to rule them all: A modular troubleshooting framework for sdn (and other) networks. In *Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research*, SOSR '15, pages 24:1–24:7, New York, NY, USA, 2015. ACM.

[226] Tim Nelson, Da Yu, Yiming Li, Rodrigo Fonseca, and Shriram Krishnamurthi. Simon: Scriptable interactive monitoring for sdns. In *Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research*, SOSR '15, pages 19:1–19:7, New York, NY, USA, 2015. ACM.

[227] Yang Xu, Yong Liu, Rahul Singh, and Shu Tao. Identifying sdn state inconsistency in openstack. In *Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research*, SOSR '15, pages 11:1–11:7, New York, NY, USA, 2015. ACM.

[228] Andrew D. Ferguson, Arjun Guha, Jordan Place, Rodrigo Fonseca, and Shriram Krishnamurthi. Participatory networking. In *Proceedings of the 2nd USENIX conference on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services*, Hot-ICE'12, pages 2–2, Berkeley, CA, USA, 2012. USENIX Association.

[229] Andrew D. Ferguson, Arjun Guha, Chen Liang, Rodrigo Fonseca, and Shriram Krishnamurthi. Hierarchical policies for software defined networks. In *Proceedings of the first workshop on Hot topics in software defined networks*, HotSDN '12, pages 37–42, New York, NY, USA, 2012. ACM.

[230] Marco Canini, Petr Kuznetsov, Dan Levin, and Stefan Schmid. Software transactional networking: Concurrent and consistent policy composition. In *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, HotSDN '13, pages 1–6, New York, NY, USA, 2013. ACM.

[231] Mark Reitblatt, Nate Foster, Jennifer Rexford, and David Walker. Consistent updates for software-defined networks: change you can believe in! In *Proceedings of the 10th ACM Workshop on Hot Topics in Networks*, HotNets-X, pages 7:1–7:6, New York, NY, USA, 2011. ACM.

[232] Mark Reitblatt, Nate Foster, Jennifer Rexford, Cole Schlesinger, and David Walker. Abstractions for network update. In *Proceedings of the ACM SIGCOMM 2012 conference on Applications, technologies, architectures, and protocols for computer communication*, SIGCOMM '12, pages 323–334, New York, NY, USA, 2012. ACM.

[233] Rick McGeer. A safe, efficient update protocol for openflow networks. In *Proceedings of the first workshop on Hot topics in software defined networks*, HotSDN '12, pages 61–66, New York, NY, USA, 2012. ACM.

[234] Soudeh Ghorbani and Matthew Caesar. Walk the line: consistent network updates with bandwidth guarantees. In *Proceedings of the first workshop on Hot topics in software defined networks*, HotSDN '12, pages 67–72, New York, NY, USA, 2012. ACM.

[235] Seyed Kaveh Fayazbakhsh, Vyas Sekar, Minlan Yu, and Jeffrey C. Mogul. Flowtags: Enforcing network-wide policies in the presence of dynamic middlebox actions. In *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, HotSDN '13, pages 19–24, New York, NY, USA, 2013. ACM.

[236] Xin Jin, Jennifer Rexford, and David Walker. Incremental update for a compositional sdn hypervisor. In *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking*, HotSDN '14, pages 187–192, New York, NY, USA, 2014. ACM.

[237] Peter Pereíni, Maciej Kuzniar, Marco Canini, and Dejan Kostić. Espres: Transparent sdn update scheduling. In *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking*, HotSDN '14, pages 73–78, New York, NY, USA, 2014. ACM.

[238] Tal Mizrahi, Efi Saat, and Yoram Moses. Timed consistent network updates. In *Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research*, SOSR '15, pages 21:1–21:14, New York, NY, USA, 2015. ACM.

[239] Masoud Moshref, Minlan Yu, and Ramesh Govindan. Resource/accuracy tradeoffs in software-defined measurement. In *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, HotSDN '13, pages 73–78, New York, NY, USA, 2013. ACM.

[240] Nikolai Matni, Ao Tang, and John C. Doyle. A case study in network architecture tradeoffs. In *Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research*, SOSR '15, pages 18:1–18:7, New York, NY, USA, 2015. ACM.

[241] Amin Tootoonchian, Monia Ghobadi, and Yashar Ganjali. Opentm: traffic matrix estimator for openflow networks. In *Proceedings of the 11th international conference on Passive and active measurement*, PAM'10, pages 201–210, Berlin, Heidelberg, Apr 2010. Springer-Verlag.

[242] Arsalan Tavakoli, Martin Casado, Teemu Koponen, and Scott Shenker. Applying nox to the data center. In *In Proceedings of ACM HotNets*, 2009.

[243] Curtis Yu, Cristian Lumezanu, Yueping Zhang, Vishal Singh, Guofei Jiang, and Harsha V. Madhyastha. Flowsense: Monitoring network utilization with zero measurement cost. In *Proceedings of Passive and Active Measurement Conference (PAM)*. PAM, 2013.

[244] Jeffrey R. Ballard, Ian Rae, and Aditya Akella. Extensible and scalable network monitoring using opensafe. In *Proceedings of the 2010 internet network management conference on Research on enterprise networking*, INM/WREN'10, pages 8–8, Berkeley, CA, USA, Apr 2010. USENIX Association.

[245] N. Grover, N. Agarwal, and K. Kataoka. liteflow: Lightweight and distributed flow monitoring platform for sdn. In *Network Softwarization (NetSoft), 2015 1st IEEE Conference on*, pages 1–9, Apr 2015.

[246] Yanlei Gong, Xiong Wang, Mehdi Malboubi, Sheng Wang, Shizhong Xu, and Chen-Nee Chuah. Towards accurate online traffic matrix estimation in software-defined networks. In *Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research*, SOSR '15, pages 26:1–26:7, New York, NY, USA, 2015. ACM.

[247] Lavanya Jose, Minlan Yu, and Jennifer Rexford. Online measurement of large traffic aggregates on commodity switches. In *Proceedings of the 11th USENIX conference on Hot topics in management of internet, cloud, and enterprise networks and services*, Hot-ICE'11, pages 13–13, Berkeley, CA, USA, 2011. USENIX Association.

[248] Mourad Soliman, Biswajit Nandy, Ioannis Lambadaris, and Peter Ashwood-Smith. Source routed forwarding with software defined control,considerations and implications, Dec 2012.

[249] Hailong Zhang and Jinyao Yan. Performance of sdn routing in comparison with legacy routing protocols. In *Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC), 2015 International Conference on*, pages 491–494, Sep 2015.

[250] Gautam Khetrapal and Saurabh Kumar Sharma. Demystifying routing services in software-defined networking, 2013. Aricent.

[251] David M. Beazley. Swig: An easy to use tool for integrating scripting languages with c and c++. In *Proceedings of the 4th Conference on USENIX Tcl/Tk Workshop, 1996 - Volume 4*, TCLTK'96, pages 15–15, Berkeley, CA, USA, 1996. USENIX Association.

[252] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. Openflow: enabling innovation in campus networks. *SIGCOMM Comput. Commun. Rev.*, 38(2):69–74, Mar 2008.

[253] Haoyu Song. Protocol-oblivious forwarding: Unleash the power of sdn through a future-proof forwarding plane. In *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, HotSDN '13, pages 127–132, New York, NY, USA, 2013. ACM.

[254] OpenStack. Openstack cloud software. `http://docs.openstack.org`, 2015.

[255] Big Switch. The open sdn architecture. `http://www.bigswitch.com`, 2012.

[256] Pankaj Berde, Matteo Gerola, Jonathan Hart, Yuta Higuchi, Masayoshi Kobayashi, Toshio Koide, Bob Lantz, Brian O'Connor, Pavlin Radoslavov, William Snow, and Guru Parulkar. Onos: Towards an open, distributed sdn os. In *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking*, HotSDN '14, pages 1–6, New York, NY, USA, 2014. ACM.

[257] Anand Krishnamurthy, Shoban P. Chandrabose, and Aaron Gember-Jacobson. Pratyaastha: An efficient elastic distributed sdn control plane. In *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking*, HotSDN '14, pages 133–138, New York, NY, USA, 2014. ACM.

[258] P. Patil, A. Gokhale, and A. Hakiri. Bootstrapping software defined network for flexible and dynamic control plane management. In *Network Softwarization (NetSoft), 2015 1st IEEE Conference on*, pages 1–5, Apr 2015.

[259] Md. Faizul Bari, Arup Raton Roy, Shihabur Rahman Chowdhury, Qi Zhang, Mohamed Faten Zhani, Reaz Ahmed, and Raouf Boutaba. Dynamic controller provisioning in software defined networks. In *9th International Conference on Network and Service Management 2013 (CNSM 2013)*, pages 18–25, Zrich, Switzerland, Oct 2013.

[260] Advait Dixit, Fang Hao, Sarit Mukherjee, T.V. Lakshman, and Ramana Kompella. Towards an elastic distributed sdn controller. In *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, HotSDN '13, pages 7–12, New York, NY, USA, 2013. ACM.

[261] Stefan Schmid and Jukka Suomela. Exploiting locality in distributed sdn control. In *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, HotSDN '13, pages 121–126, New York, NY, USA, 2013. ACM.

[262] Xuan-Nam Nguyen, Damien Saucez, Chadi Barakat, and Thierry Turletti. Optimizing rules placement in openflow networks: Trading routing for better efficiency. In *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking*, HotSDN '14, pages 127–132, New York, NY, USA, 2014. ACM.

[263] Naga Katta, Omid Alipourfard, Jennifer Rexford, and David Walker. Infinite cacheflow in software-defined networks. In *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking*, HotSDN '14, pages 175–180, New York, NY, USA, 2014. ACM.

[264] Guohan Lu, Rui Miao, Yongqiang Xiong, and Chuanxiong Guo. Using cpu as a traffic co-processing unit in commodity switches. In *Proceedings of the first workshop on Hot topics in software defined networks*, HotSDN '12, pages 31–36, New York, NY, USA, 2012. ACM.

[265] Huynh Tu Dang, Daniele Sciascia, Marco Canini, Fernando Pedone, and Robert Soulé. Netpaxos: Consensus at network speed. In *Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research*, SOSR '15, pages 5:1–5:7, New York, NY, USA, 2015. ACM.

[266] Naga Katta, Haoyu Zhang, Michael Freedman, and Jennifer Rexford. Ravana: Controller fault-tolerance in software-defined networking. In *Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research*, SOSR '15, pages 4:1–4:12, New York, NY, USA, 2015. ACM.

[267] Michael Borokhovich, Liron Schiff, and Stefan Schmid. Provable data plane connectivity with local fast failover: Introducing openflow graph algorithms. In *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking*, HotSDN '14, pages 121–126, New York, NY, USA, 2014. ACM.

[268] Maulik Desai and Thyagarajan Nandagopal. Coping with link failures in centralized control plane architectures. In *Proceedings of the 2Nd International Conference on COMmunication Systems and NETworks*, COMSNETS'10, pages 79–88, Piscataway, NJ, USA, 2010. IEEE Press.

[269] S. Sharma, D. Staessens, D. Colle, M. Pickavet, and P. Demeester. Enabling fast failure recovery in openflow networks. In *Design of Reliable Communication Networks (DRCN), 2011 8th International Workshop on the*, pages 164–171, Oct 2011.

[270] S. Sharma, D. Staessens, D. Colle, M. Pickavet, and P. Demeester. Fast failure recovery for in-band openflow networks. In *Design of Reliable Communication Networks (DRCN), 2013 9th International Conference on the*, pages 52–59, Mar 2013.

[271] N. M. Sahri and Koji Okamura. Fast failover mechanism for software defined networking: Openflow based. In *Proceedings of The Ninth International Conference on Future Internet Technologies*, CFI '14, pages 16:1–16:2, New York, NY, USA, 2014. ACM.

[272] Antonio Capone, Carmelo Cascone, Alessandro Q. T. Nguyen, and Brunilde Sansò. Detour planning for fast and reliable failure recovery in SDN with openstate. *CoRR*, abs/1411.7711, 2014.

[273] Sushant Jain, Alok Kumar, Subhasree Mandal, Joon Ong, Leon Poutievski, Arjun Singh, Subbaiah Venkata, Jim Wanderer, Junlan Zhou, Min Zhu, Jon Zolla, Urs Hölzle, Stephen Stuart, and Amin Vahdat. B4: Experience with a globally-deployed software defined wan. *SIGCOMM Comput. Commun. Rev.*, 43(4):3–14, Aug 2013.

[274] Elias Molina, Eduardo Jacob, Jon Matias, Naiara Moreira, and Armando Astarloa. Availability improvement of layer 2 seamless networks using openflow. *The Scientific World Journal*, 2015, 2015.

[275] Dongkyun Kim, Won-Hyeak Lee, Myung-Il Kim, and Seung-Hae Kim. Dvnoc: Multi-domain sdn operations and management framework. In *Research Notes in Information Science (RNIS)*, volume 14, pages 540–543. Advanced Institute of Convergence Information Technology (AICIT), 2013.

[276] ukasz Podleski, Radek Krzywania, Miosz Przywecki, Eduardo Jacob, Alaitz Mendiola, Ignacio Jos, Albert Aznar-Baranda, Vico-Oton, Kurt Baumann, and Christos Argyropoulos. Multi-domain software defined network: exploring possibilities. In *TERENA Networking Conference 2014 (TNC2014)*. Trans-European Research and Education Networking Association (TERENA), 2014.

[277] Boyang Zhou, Haifeng Zhou, Wen Gao, Xiaoyan Hong, Bin Wang, and Chunming Wu. Lossless reconfiguration protocol for multi-domain data plane in software-defined networks. In *Computer Communications Workshops (INFOCOM WKSHPS), 2014 IEEE Conference on*, pages 193–194, Apr 2014.

[278] S. Civanlar, E. Lokman, B. Kaytaz, and A. Murat Tekalp. Distributed management of service-enabled flow-paths across multiple sdn domains. In *Networks and Communications (EuCNC), 2015 European Conference on*, pages 360–364, Jun 2015.

[279] Dan Marconett and S.J.B. Yoo. Flowbroker: A software-defined network controller architecture for multi-domain brokering and reputation. *Journal of Network and Systems Management*, 23(2):328–359, 2015.

[280] Jun-Huy Lam, Sang-Gon Lee, Hoon-Jae Lee, and Y.E. Oktian. Securing distributed sdn with ibc. In *Ubiquitous and Future Networks (ICUFN), 2015 Seventh International Conference on*, pages 921–925, Jul 2015.

[281] Rodrigo Braga, Edjard Mota, and Alexandre Passito. Lightweight ddos flooding attack detection using nox/openflow. In *Local Computer Networks (LCN), 2010 IEEE 35th Conference on*, pages 408 –415, Denver, CO, USA, Oct 2010. IEEE.

[282] Manar Jammal, Taranpreet Singh, Abdallah Shami, Rasool Asal, and Yiming Li. Software-defined networking: State of the art and research challenges. *CoRR*, abs/1406.0124, 2014.

[283] Natali Ruchansky and Davide Proserpio. A (not) nice way to verify the openflow switch specification: Formal modelling of the openflow switch using alloy. *SIGCOMM Comput. Commun. Rev.*, 43(4):527–528, Aug 2013.

[284] Ehab Al-Shaer and Saeed Al-Haj. Flowchecker: configuration analysis and verification of federated openflow infrastructures. In *Proceedings of the 3rd ACM workshop on Assurable and usable security configuration*, SafeConfig '10, pages 37–44, New York, NY, USA, 2010. ACM.

[285] OpenFlow Switch Consortium. Openflow switch consortium. `http://www.OpenFlowSwitch.org`, 2008.

[286] OpenDayLight. Opendaylight. `http://www.opendaylight.org`, 2013.

[287] B. Nunes, M. Mendonca, X. Nguyen, K. Obraczka, and T. Turletti. A survey of software-defined networking: Past, present, and future of programmable networks. *Communications Surveys Tutorials, IEEE*, PP(99):1–18, 2014.

[288] Juniper Networks. 5 tenets of instant evolution. http://www.juniper.net/instantevolution, Aug 2015.

[289] IBM Systems and Technology Group. Openflow: The next generation in networking interoperability, May 2011.

[290] Glen Hunt. Software defined networking: An imperative for the service provider network. http://www.juniper.net, May 2013.

[291] Sangam Racherla, David Cain, Scott Irwin, Per Ljungstrm, Pushkar Patil, and Alessio M. Tarenzio. Implementing ibm software defined network for virtual environments, Sep 2014.

[292] Oracle. Oracle sdn performance acceleration with software-defined networking. http://www.oracle.com, 2015.

[293] Cisco. Cisco open network environment: Bring the network closer to applications. http://www.cisco.com, 2015.

[294] Pica8. Pica8: White box sdn. http://www.pica8.com, 2015.

[295] Pingping Lin, Jonathan Hart, Umesh Krishnaswamy, Tetsuya Murakami, Masayoshi Kobayashi, Ali Al-Shabibi, Kuang-Ching Wang, and Jun Bi. Seamless interworking of sdn and ip. *SIGCOMM Comput. Commun. Rev.*, 43(4):475–476, Aug 2013.

[296] MetaSwitch. The network software provider: A new vendor category. http://www.metaswitch.com, 2015.

[297] Ericsson. The real-time cloud: Combining cloud, nfv, and service provider sdn. http://www.ericsson.com, Feb 2014.

[298] Ericsson. Virtual cpe and software defined networking. http://www.ericsson.com, Mar 2014.

[299] Ericsson. Dynamic service chaining with sdn. http://www.ericsson.com, May 2014.

[300] Joe Skorupa, Mark Fabbi, and Akshay K. Sharma. Ending the confusion about software-defined networking: A taxonomy, Mar 2013.

[301] Ryan Hand and Eric Keller. Closedflow: Openflow-like control over proprietary devices. In *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking*, HotSDN '14, pages 7–12, New York, NY, USA, 2014. ACM.

[302] C. Jasson Casey, Andrew Sutton, and Alex Sprintson. tinynbi: Distilling an api from essential openflow abstractions. In *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking*, HotSDN '14, pages 37–42, New York, NY, USA, 2014. ACM.

[303] S.J. Vaughan-Nichols. Openflow: The next generation of the network? *Computer*, 44(8):13–15, Aug 2011.

[304] Juniper Networks. Decoding software defined networking. http://www.juniper.net, Feb 2013.

[305] Zeus Kerravala. Cisco intelligent wan is the foundation for the software-defined wan. `http://www.cisco.com`, Sep 2015.

[306] Jennifer Rexford, Albert Greenberg, Gisli Hjalmtysson, David A. Maltz, Andy Myers, Geoffrey Xie, Jibin Zhan, and Hui Zhang. Network-wide decision making: Toward a wafer-thin control plane. In *In Proceedings of HotNets III*, 2004.

[307] Ossama Younis and Sonia Fahmy. Constraint-based routing in the internet: Basic principles and recent research. *IEEE Communications Surveys and Tutorials*, 5:2–13, 2003.

[308] S. Shenker and C. Partridge. Specification of guaranteed quality of service. `http://www.ietf.org`, 1997. RFC 2212.

[309] J. Wroclawski. Specification of the controlled-load network element service. `http://www.ietf.org`, 1997. RFC 2211.

[310] Puqi Perry Tang and Tsung-Yuan Charles Tai. Network traffic characterization using token bucket model. In *In Proceedings of IEEE Infocom99*, pages 51–62, 1999.

[311] Rajiv Chakravorty, Subrat Kar, and Peyman Farjami. End-to-end internet quality of service (qos): An overview of issues, architectures and frameworks. In *Proceedings of International Conference on Information Technology (ICIT)*, 2000.

[312] Jos Ruela and Manuel Ricardo. MPLS - Multiprotocol Label Switching. In *The Industrial Information Technology Handbook*, pages 1–9. CRC Press, 2004.

[313] George F. Riley and Thomas R. Henderson. The ns-3 network simulator modeling and tools for network simulation. In *Modeling and Tools for Network Simulation*, chapter 2, pages 15–34. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[314] Johannes Schneider and Roger Wattenhofer. Trading bit, message, and time complexity of distributed algorithms. In *25th International Symposium on Distributed Computing (DISC), Rome, Italy*, Sep 2011.

[315] P. Ashwood-Smith and M. Soliman. Sdn state reduction. `http://tools.ietf.org`, Jul 2013.

[316] Cisco. Cisco visual networking index: Forecast and methodology, 2012 - 2017. `http://www.cisco.com`, 2012.

[317] Ted H. Szymanski and Dave Gilbert. Provisioning mission-critical telerobotic control systems over internet backbone networks with essentially-perfect qos. *IEEE J.Sel. A. Commun.*, 28(5):630–643, Jun 2010.

[318] Costin Raiciu, Christoph Paasch, Sebastien Barre, Alan Ford, Michio Honda, Fabien Duchene, Olivier Bonaventure, and Mark Handley. How hard can it be? designing and implementing a deployable multipath tcp. In *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation*, NSDI'12, pages 29–29, Berkeley, CA, USA, 2012. USENIX Association.

[319] Gang Liu and K. G. Ramakrishnan. A*Prune: An Algorithm for Finding K Shortest Paths Subject to Multiple Constraints. In *IEEE INFOCOM*, volume 2, pages 743–749, 2001.

[320] The ns-3 network simulator. http://www.nsnam.org, 2015.

[321] Radivoj Petrovic and S. Jovanovic. Two algorithms for determining the most reliable path of a network. *Reliability, IEEE Transactions on*, R-28(2):115–119, 1979.

[322] Do-Hyeon Lee, Doo-Young Kim, Jae-Il Jung, and Young-Soo An. An algorithm for acquiring reliable path in abnormal traffic condition. In *Proceedings of the 2008 International Conference on Convergence and Hybrid Information Technology*, ICHIT '08, pages 682–686. IEEE Computer Society, 2008.

[323] Jing Wang, Jianming Zhu, and Haoxiong Yang. Reliable path selection problem in uncertain traffic network after natural disaster. *Mathematical Problems in Engineering*, 2013, 2013. 5 pages.

[324] J. Gozdecki, A. Jajszczyk, and R. Stankiewicz. Quality of service terminology in ip networks. *Communications Magazine, IEEE*, 41(3):153 – 159, Mar 2003.

[325] Y. Rekhter, T. Li, and S. Hares. Rfc 4271: A border gateway protocol 4 (bgp-4), 2006.

[326] Wes Simpson. *Video Over IP: IPTV, Internet Video, H.264, P2P, Web TV, and Streaming: A Complete Guide to Understanding the Technology*. Focal Press, 2008.

[327] OpenDaylight. Odl-sdni app (opendaylight- sdn interface application). `https://wiki.opendaylight.org/view/Project_Proposals:ODL-SDNi_App`, Jun 2015.

[328] Deepankar Gupta and Rafat Jahan. Inter-sdn controller communication: Using border gateway protocol. `http://www.tcs.com`, 2014.

[329] GAO Wen, ZHOU Boyang, WU Chunming, ZHOU Haifeng, JIANG Ming, and HONG Xiaoyan. Safe reconfiguring data plane via supervision over resource and flow states. *Chinese Journal of Electronics*, 24(CJE-3):642, 2015.

[330] R. Guerin and A. Orda. Qos based routing in networks with inaccurate information: theory and algorithms. In *INFOCOM '97. Sixteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Driving the Information Revolution., Proceedings IEEE*, volume 1, pages 75–83 vol.1, Apr 1997.

[331] William B. Norton. *The Internet Peering Playbook: Connecting to the Core of the Internet*. DrPeering Press, 2014.

[332] Google. Hangouts system requirements on a computer. `https://support.google.com`, 2015.

[333] Microsoft. How much bandwidth does skype need? `https://support.skype.com`, 2015.

[334] HP. Hp van sdn controller 2.4 installation guide. `http://www.hp.com`, 2014.

[335] HP. Hp van sdn controller 2.4 administrator guide. `http://www.hp.com`, 2014.

[336] HP. Hp van sdn controller programming guide. `http://www.hp.com`, 2013.

[337] HP. Hp sdn controller architecture. `http://www.hp.com`, 2013.

[338] HP. Hp switch software management and configuration guide for wb.15.16. `http://www.hp.com`, 2013.

[339] J. Rubio-Loyola, A. Galis, A. Astorga, J. Serrat, L. Lefevre, A. Fischer, A. Paler, and H. Meer. Scalable service deployment on software-defined networks. *Communications Magazine, IEEE*, 49(12):84 –93, Dec 2011.

[340] Frank Drr. Towards cloud-assisted software-defined networking. Universitat Stuttgart Technical Report TR-2012-04, Institute of Parallel and Distributed Systems Universitat Stuttgart, 70569 Stuttgart Germany, Aug 2012.

[341] Brandon Heller, Srini Seetharaman, Priya Mahadevan, Yiannis Yiakoumis, Puneet Sharma, Sujata Banerjee, and Nick McKeown. Elastictree: saving energy in data center networks. In *Proceedings of the 7th USENIX conference on Networked systems design and implementation*, NSDI'10, pages 17–17, Berkeley, CA, USA, 2010. USENIX Association.

[342] Zdravko Bozakov. An open router virtualization framework using a programmable forwarding plane. In *Proceedings of the ACM SIGCOMM 2010 conference*, SIGCOMM '10, pages 439–440, New York, NY, USA, 2010.

[343] Raj Jain and Subharthi Paul. Openadn: mobile apps on global clouds using software defined networking. In *Proceedings of the third ACM workshop on Mobile cloud computing and services*, MCS '12, pages 1–2, New York, NY, USA, 2012. ACM.

[344] Kwangtae Jeong, Jinwook Kim, and Young-Tak Kim. Qos-aware network operating system for software defined networking with generalized openflows. In *NOMS*, pages 1167–1174. IEEE, 2012.

[345] Marcelo Ribeiro Nascimento, Christian Esteve Rothenberg, Marcos Rogério Salvador, and Maurício Ferreira Magalhães. Quagflow: partnering quagga with openflow. *SIGCOMM Comput. Commun. Rev.*, 40(4):441–442, Aug 2010.

[346] Marcelo R. Nascimento, Christian E. Rothenberg, Marcos R. Salvador, Carlos N. A. Corrêa, Sidney C. de Lucena, and Maurício F. Magalhães. Virtual routers as a service: the routeflow approach leveraging software-defined networks. In *Proceedings of the 6th International Conference on Future Internet Technologies*, CFI '11, pages 34–37, New York, NY, USA, 2011. ACM.

APPENDICES

Appendix A Data Tables

The data set used to generate Figure 3.7.

Requests	Set	Unset	Request	Accept	Remove	Error	Total
500	2004	2000	2000	3000	3000	0	12005
1000	4004	4000	4000	6002	6000	0	24007
2000	8004	8000	8000	12000	12000	0	48005
3000	12004	12000	12000	18000	18000	0	72005
3500	14004	14000	14000	21000	21000	0	84005
4000	16004	16000	16000	24000	24000	0	96005

The data set used to generate Figure 3.8.

Requests	Set	Unset	Request	Accept	Remove	Error	Total
500	2505	2500	2500	3498	3500	0	14504
1000	5005	5000	5000	7000	7000	0	29006
2000	10005	10000	10000	14000	14000	0	58006
3000	15005	15000	15000	20999	21000	0	87005
3500	17505	17500	17500	24495	24500	0	101501

The data set used to generate Figure 5.4.

Reliability	RVSDN	MPLS	VSDN
0.90	103333	33333	103333
0.91	103333	33333	103333
0.92	103333	33333	103333
0.93	103333	33333	103333
0.94	103333	33333	103333
0.95	103333	33333	103333
0.96	103333	33333	103333
0.97	103333	33333	103333
0.98	103333	33333	70000
0.99	103333	33333	36666
0.993	103333	33333	36666
0.995	103333	3333	3333
0.996	103333	3333	3333
0.997	103333	3333	3333
0.998	103333	3333	3333
0.999	103333	3333	3333

The data set used to generate Figure 4.2 and Figure 4.3.

Requests	Set	Unset	Remove-control	Accept-control	Set-data	Unset-data	Request-data	Remove-data	Accept-data	Control plane	Data plane	Total
500	3507	3500	500	500	2004	2000	2000	2500	2500	8008	11005	19014
1000	7007	7000	1000	1001	4004	4000	4000	5000	4998	16009	22003	38013
1500	10507	10500	1500	1506	6004	6000	6000	7500	7488	24014	32993	57008
2000	14007	14000	2000	2021	8004	8000	8000	10000	9961	32029	43966	75996
2500	17507	17500	2500	2509	10004	10000	10000	12500	12482	40017	54987	95005
3000	21007	21000	3000	3016	12004	12000	12000	15000	14968	48024	65973	113998
3500	24507	24500	3500	3512	14004	14000	14000	17500	17476	56020	76981	133002
4000	28007	28000	4000	4064	16004	16000	16000	20000	19876	64072	87881	151954
4500	31507	31500	4500	4520	18004	18000	18000	22500	22460	72028	98965	170994
5000	35007	35000	5000	5021	20004	20000	20000	25000	24958	80029	109963	189993
5500	38507	38500	5500	5524	22004	22000	22000	27500	27452	88032	120957	208990
6000	42007	42000	6000	6026	24004	24000	24000	30000	29948	96034	131953	227988
6500	45507	45500	6500	6525	26004	26000	26000	32500	32450	104033	142955	246989
7000	49007	49000	7000	7042	28004	28000	28000	35000	34917	112050	153922	265973
7500	52507	52500	7500	7514	30004	30000	30000	37500	37475	120022	164980	285003
8000	56007	56000	8000	8025	32004	32000	32000	40000	39952	128033	175957	303991

The data set used to generate Figure 4.2 and Figure 4.3.

Requests	Set	Unset	Remove-control	Accept-control	Set-data	Unset-data	Request-data	Remove-data	Accept-data	Control plane	Data plane	Total
500	1002	1000	500	500	2505	4000	2000	2500	2500	3003	13506	16510
1000	2002	2000	1000	1000	5005	8000	4000	5000	5000	6003	27006	33010
1500	3002	3000	1500	1500	7505	12000	6000	6645	7500	9003	39651	48655
2000	4002	2310	1156	2000	10005	8674	8000	7717	10000	9469	44397	53867
2500	5002	2280	1141	2500	12505	8386	10000	7386	12500	10924	50778	61703
3000	6002	4748	2375	3000	15005	18457	12000	15447	15000	16126	75910	92037
3500	7002	7000	3500	3500	17505	28000	14000	12061	17500	21003	89067	110071
4000	8002	8000	4000	4000	20005	32000	16000	19895	20000	24003	107901	131905
4500	9002	9000	4500	4500	22505	36000	18000	22501	22500	27003	121507	148511
5000	10002	10000	5000	5000	25005	40000	20000	24994	25000	30003	135000	165004
5500	11002	11000	5500	5500	27505	44000	22000	27500	27500	33003	148506	181510
6000	12002	12000	6000	6000	30005	48000	24000	30007	30000	36003	162013	198017
6500	13002	13000	6500	6500	32505	52000	26000	32500	32500	39003	175506	214510
7000	14002	14000	7000	7000	35005	56000	28000	34997	35000	42003	189003	231007
7500	15002	15000	7500	7500	37505	60000	30000	37504	37500	45003	202510	247514
8000	16002	16000	8000	8000	40005	64000	32000	40000	40000	48003	216006	264010

The data set used to generate Figure 4.4 and Figure 4.5.

Requests	Set	Unset	Remove-control	Accept-control	Set-data	Unset-data	Request-data	Remove-data	Accept-data	Control plane	Data plane	Total
500	7014	7000	500	501	2505	2500	2500	3000	2997	15016	13503	28520
1000	14014	14000	1000	1010	5005	5000	5000	6000	5970	30025	26976	57002
1500	21014	21000	1500	1507	7505	7500	7500	9000	8979	5022	40485	85508
2000	28014	28000	2000	2008	10005	10000	10000	12000	11976	60023	53982	114006
2500	35014	35000	2500	2517	12505	12500	12500	15000	14952	75032	67458	142491
3000	42014	42000	3000	3018	15005	15000	15000	18000	17949	90033	80955	170989
3500	49014	49000	3500	3516	17505	17500	17500	21000	20952	105031	94458	199490
4000	56014	56000	4000	4024	20005	20000	20000	24000	23928	120039	107934	227974
4500	63014	63000	4500	4538	22505	22500	22500	27000	26886	135053	121392	256446
5000	70014	70000	5000	5065	25005	25000	25000	30002	29805	150080	134813	284894
5500	77014	77000	5500	5537	27505	27500	27500	33000	32894	165052	148400	313453
6000	84014	84000	6000	6067	30005	30000	30000	36000	35801	180082	161807	341890
6500	91014	91000	6500	6561	32505	32500	32500	39000	38817	195076	175323	370400
7000	98014	98000	7000	7005	35005	35000	35000	42000	41991	210020	188997	399018
7500	105014	105000	7500	7560	37505	37500	37500	45000	44824	225075	202330	427406
8000	112014	112000	8000	8054	40005	40000	40000	48000	47838	240069	215844	455914

The data set used to generate Figure 4.4 and Figure 4.5.

Requests	Set	Unset	Remove-control	Accept-control	Set-data	Unset-data	Request-data	Remove-data	Accept-data	Control plane	Data plane	Total
500	1002	1000	500	500	3006	5000	2500	3000	3000	3003	16507	19511
1000	2002	2000	1000	1000	6006	10000	5000	6000	6000	6003	33007	39011
1500	3002	3000	1500	1500	9006	15000	7500	9002	9000	9003	49509	58513
2000	4002	4000	2000	2000	12006	20000	10000	12000	12000	12003	66007	78011
2500	5002	5000	2500	2500	15006	25000	12500	15004	15000	15003	82511	97515
3000	6002	6000	3000	3000	18006	30000	15000	18000	18000	18003	99007	117011
3500	7002	7000	3500	3500	21006	35000	17500	21002	21000	21003	115509	136513
4000	8002	8000	4000	4000	24006	40000	20000	24000	24000	24003	132007	156011
4500	9002	9000	4500	4500	27006	45000	22500	26996	27000	27003	148503	175507
5000	10002	10000	5000	5000	30006	50000	25000	30000	30000	30003	165007	195011
5500	11002	11000	5500	5500	33006	55000	27500	33000	33000	33003	181507	214511
6000	12002	12000	6000	6000	36006	60000	30000	35900	36000	36003	197907	233911
6500	13002	13000	6500	6500	39006	65000	32500	39000	39000	39003	214507	253511
7000	14002	14000	7000	7000	42006	70000	35000	42018	42000	42003	231025	273029
7500	15002	15000	7500	7500	45006	75000	37500	45010	45000	45003	247517	292521
8000	16002	16000	8000	8000	48006	80000	40000	48000	48000	48003	264007	312011

The data set used to generate Figure 6.7.

Requests	Request	Accept	Remove	Controller-request	Controller-accept	Total
50	400	400	400	100	100	1400
100	800	800	800	200	200	2800
150	1200	1200	1200	300	300	4200
200	1600	1600	1600	400	400	5600
250	2000	2000	2000	500	500	7000
300	2400	2400	2400	600	600	8400
350	2800	2800	2800	700	700	9800
400	3200	3200	3200	800	800	11200
450	3600	3600	3600	900	900	12600
500	4000	4000	4000	1000	1000	14000

Appendix B Network Topologies

The network topology used to generate Figure 4.2 and Figure 4.3.

```
void initialize6NodeTopology()
{
  // addEdge(source, target, bandWidth, delay, jitter, name)
  graph.addEdge(A, B, 40000, 5.5f, 5, "A-B");
  graph.addEdge(B, C, 40000, 5.0f, 8, "B-Edge-C");
  graph.addEdge(B, E, 40000, 8.0f, 7, "B-E");
  graph.addEdge(D, E, 40000, 7.0f, 5, "D-E");
  graph.addEdge(A, D, 40000, 5.0f, 5, "A-D");
  graph.addEdge(C, E, 40000, 6.0f, 5, "Edge-C-E");
  graph.addEdge(F, A, 40000, 5.0f, 5, "Edge-F-A");
  graph.addEdge(F, D, 40000, 2.0f, 5, "Edge-F-D");
}
```

The network topology used to generate Figure 4.4 and Figure 4.5.

```
void initialize13NodeTopology()
{
  // addEdge(source, target, bandWidth, delay, jitter, name)
  graph.addEdge(A, B, 40000, 5.5f, 1, "Edge-A-B");
  graph.addEdge(A, C, 40000, 5.5f, 1, "Edge-A-C");
  graph.addEdge(A, D, 40000, 5.5f, 1, "Edge-A-D");

  graph.addEdge(B, E, 40000, 5.0f, 1, "B-E");
  graph.addEdge(B, C, 40000, 5.0f, 1, "B-C");
  graph.addEdge(B, F, 40000, 5.0f, 1, "B-F");

  graph.addEdge(C, F, 40000, 5.0f, 1, "C-F");
  graph.addEdge(C, D, 40000, 5.0f, 1, "C-D");
```

```
    graph.addEdge(C, G, 40000, 5.0f, 1, "C-G");

    graph.addEdge(D, G, 40000, 5.0f, 1, "D-G");

    graph.addEdge(E, H, 40000, 5.0f, 1, "E-H");
    graph.addEdge(F, I, 40000, 5.0f, 1, "F-I");
    graph.addEdge(G, J, 40000, 5.0f, 1, "G-J");

    graph.addEdge(E, K, 40000, 8.0f, 1, "E-K");
    graph.addEdge(K, I, 40000, 7.0f, 1, "K-I");
    graph.addEdge(F, K, 40000, 5.0f, 1, "F-K");
    graph.addEdge(K, H, 40000, 5.0f, 1, "K-H");

    graph.addEdge(M, F, 40000, 8.0f, 1, "M-F");
    graph.addEdge(M, G, 40000, 7.0f, 1, "M-G");
    graph.addEdge(M, J, 40000, 5.0f, 1, "M-J");
    graph.addEdge(M, I, 40000, 5.0f, 1, "M-I");

    graph.addEdge(L, H, 40000, 6.0f, 1, "Edge-L-H");
    graph.addEdge(L, I, 40000, 5.0f, 1, "Edge-L-I");
    graph.addEdge(L, J, 40000, 2.0f, 1, "Edge-L-J");
}
```

Appendix C SDN Lab Experience

My plan was to develop VSDN on the HP VAN Controller. The lab had one HP 2900 series Openflow switch that connected two application servers and one HP VAN controller that I installed [334, 335]. HP gives the developer a 60-day trial license for the HP VAN controller which has 50-node limit. The HP VAN controller SDK [336] is substantial and was new at the time of research. The challenging aspect of the research was mapping the concepts of the VSDN architecture into the HP SDN controller and switch architecture [337]. For example, the topology monitor of VSDN with the topology monitor of the HP VAN controller. The HP VAN controller targets enterprise-ready networking applications, requiring a collaboration and development effort from a team of researchers. The configuration of the HP switch [338] was challenging, learning the commands and how the HP switch behaves. A major challenge was configuring the Linux operating system and ensuring the correct dependencies were installed on the application servers, HP VAN controller, and switch.

VITA

VITA

HAROLD OWENS II
Department of Computer and Information Science
Purdue University
Email: owensh@iupui.edu

EDUCATION

December 2016 Purdue University Indianapolis, IN

Major: Computer Science Advisor: Arjan Durresi

- Doctor of Philosophy
- Research Area: Area of focus is a combination of software engineering, distributed systems, and computer networks. My dissertation research focuses on the design and development of Software-Defined Networking (SDN) architecture and applications such as routing, quality of service (QoS), and traffic engineering
- Dissertation: Provisioning End-to-End Quality of Service for Real-time Interactive Video over Software-Defined Networking

Relevant Coursework

- Data Communication and Computer Networks, Cloud Computing, Programming Languages, Computer Algorithms, Distributed Systems, Advance Distributed Systems, Database Systems, Object-Oriented Design and Programming, and Software Engineering

August 2003 University of North Texas Denton, TX

Major: Computer Science

- Master of Science
- Research Area: Adhoc and Wireless Networks
- Thesis: Resource Allocations in Wireless and Mobile Networks

Relevant Coursework

- Human-Computer Interactions, Artificial Intelligence, Mobile Ad-hoc Networking, Computer Architecture, Parallel Programming, Computer Algorithms, Operating System Design, and Programming Languages

December 1999 McMurry University Abilene, TX

Major: Computer Science Minor: Mathematics

- Bachelor of Science
- 28 Mathematics Credit Hours

May 2000 Community College of the Air Force Maxwell AFB, AL

Major: Avionics Systems Technology

- Associate of Applied Science

PROFESSIONAL EXPERIENCE

2016 – Present	Interactive Intelligence Inc.	Indianapolis, IN

Director of Testing and Development

2013 – 2016	Interactive Intelligence Inc.	Indianapolis, IN

Senior Manager of Testing and Development

2011 – 2013	Interactive Intelligence Inc.	Indianapolis, IN

Manager of Testing and Development

2011 – 2013	IUPUI	Indianapolis, IN

Graduate Teaching Assistant (TA)/Research Assistant (RA), SE-DRC Group

2011 – 2011	Intel Corporation	Portland, OR

Graduate Internship Technical, Intel Architecture Group, PC Client Group (PCCG)

2003 - 2010	Interactive Intelligence Inc.	Indianapolis, IN

Senior Software Engineer

2001 - 2002	Aastra	Addison, TX

Software Engineer

PUBLICATIONS

Refereed Journals

1. Owens II, H, and Durresi, A., (2015, October), Video over Software-Defined Networking (VSDN), Computer Networks (September 2015), http://dx.doi.org/10.1016/j.comnet.2015.09.009.
2. Owens II, H. and Boukerche, A., Media synchronization and QoS packet scheduling algorithms for wireless systems, ACM, Mobile Networks and Applications (February 2005), Volume 10 Issue 1-2.

Selective Conferences

1. Owens II, H. and Durresi, A., Explicit Routing in Software-Defined Networking (ERSDN): Addressing Controller Scalability, 17th International Conference on Network-Based Information Systems, NBiS 2014, p.128-134, Salerno, Italy, September 10-12, 2014.
2. Owens II, H., Durresi, A., and Jain, R., Reliable Video over Software-Defined Networking (RVSDN), Global Communications Conference (GLOBECOM), p. 1974-1979, 2014 IEEE.
3. Owens II, H. and Durresi, A., Video over Software-Defined Networking (VSDN), 16th International Conference on Network-Based Information Systems (NBiS), Gwangju, Korea, September, 2013.
4. Owens II, H. and Hill, J. H., Generating Valid Interface Definition Language from Succinct Models, Proceeding of the 14th IEEE Int'l Symposium on Object/Component/Service-oriented Real-time Distributed Computing (ISORC 2011). Newport Beach, CA, USA. March 2011.

5. Owens II, H. and Boukerche, A., Energy Aware Routing Protocol for Mobile and Wireless Ad hoc Networks, Proceedings of the 28th Annual IEEE International Conference on Local Computer Networks, p.768, October 20-24, 2003.
6. Owens II, H. and Boukerche, A., "Media Synchronization Quality of Packet Scheduling Algorithm for Wireless Systems using Hard Handoff", IEEE MWCM, Singapore, 2003.
7. Owens II, H. and Boukerche, A., "Media Synchronization Quality of Packet Scheduling Algorithm for Wireless Systems using Soft Handoff", Proceedings of the 11th IEEE/ACM International Symposium on Modeling Analysis and Simulation of Computer Telecommunications Systems, IEEE MASCOTS, USA 2003.

Submitted Papers

1. Owens II, H. and Durresi, A., Multi-Domain Video over Software-Defined Networking (MDVSDN), The 31st IEEE International Conference on Advanced Information Networking and Applications (AINA-2017) Tamkang University, Taipei, Taiwan, 2017.
2. Owens II, H. and Durresi, A., Software-Defined Networking Survey: A Research Landscape, Journal of Network and Computer Applications - Elsevier, 2016.

Papers in Preparation

1. Owens II, H. and Durresi, A., Trusted Video over Software-Defined Networking (TVSDN), 2016.

Poster Presentations

1. Owens II, H. and Durresi, A., Video over Software-Defined Networking (VSDN), IUPUI Research Day, Indianapolis, IN, April 5, 2013.
2. Owens II, H. and Durresi, A., Cloud Computing: Applications, Benefits, and Challenges, 2011 Research Conference of the Midwest Crossroads Alliance for Graduate Education and the Professoriate (AGEP) at IUPUI on November 4-6, 2011.

Workshop

1. Hill, J.H. and Owens II, H. (2011, May). Towards Using Abstract Behavior Models to Evaluate Software System Performance Properties. 5th International Research Workshop on Advances and Innovations in Software Testing, Memphis, TN.

PROFESSIONAL SERVICE

External reviews

ISRCS 2015, IEEE SCC 2016

Journal reviews

COMNET 2015, COMNET 2016

TECHNICAL EXPERTISE

- Software Design Patterns: Architectural Patterns, OO Design Patterns, Anti-Patterns, and SOA Design Patterns
- Software Development Lifecycle Processes: SCRUM, Agile, Extreme Programming (XP), and iterative
- Programming Standards and Languages: C/C++, Java, C#, .NET, ASP .net, MVC, Windows Forms, WPF, WCF, OOA/OOD/OOP, DCOM, COBRA, UML, Assembly Language, Visual Basic, Unix Shell, Perl, and SQL
- Text Based Data Formats and Manipulation: XML, JSON, XSLT, and regex
- Web Technologies: Bootstrap, AngularJS, REST, SOAP, HTML, XML, WSDL, HTTP, CSS, and JavaScript
- Operating Systems Platforms: Windows, LINUX, UNIX, and Mac OS
- Programming Applications: Microsoft Visual Studio, GNU C/C++, SlickEdit, and Eclipse
- Database Applications: MySQL, SQL Server, and Oracle

IT CERTIFICATION TRAINING

- **AWS Certified Solutions Architect – Associate Certification,** Amazon, 2015
- **A+ Certification**, CompTIA, Chicago, IL, 2006
- **Network+ Certification**, CompTIA, Chicago, IL, 2006
- **An Effective Introduction to the STL,** Developmentor Inc., FW291-10, 2006
- **High-Performance C++ Programming**, Developmentor Inc., FW295-10, 2006

HONORS AND AWARDS

- Kappa Mu Epsilon: Mathematics Honor Society
- NSF GK-12 Fellowship Recipient
- GEM Fellowship Recipient
- SREB Doctoral Scholar
- O.P. Thrane Scholarship
- Outstanding Junior Award

PROFESSIONAL SOCIETIES

- McMurry University Alumni Association
- University of North Texas Alumni Association
- Institute of Electrical and Electronics Engineers (IEEE)
- Association for Computing Machinery (ACM)